YAMAHA
YAMAHA MOTOR CO., LTD.

MRC SERIES
QRC SERIES

PROGRAMMING MANUAL

Programming MANUAL

Contents

1 The Yamaha Robot Language ---------- 1

2 Characters ---------- 2

3 Program names ---------- 3

4 Identifiers ---------- 7

5 Command Statement Format ---------- 8

6 Numerals ---------- 9

6-1 Character Type Numerals 9

6-2 Value Type Numerals 9

6-2-1 Integer Type Numerals 9

6-2-2 Real Number Type Numerals 10

7 Variables ---------- 11

7-1 Valid Range of Variables 11

7-1-1 Valid Range of Dynamic Variables 11

7-1-2 Valid Range of Static Variables 11

7-2 Types of Variables 12

7-3 Array Variables 12

7-4 Clearing Variables 13

7-4-1 Clearing Dynamic Variables 13

7-4-2 Clearing Static Variables 14

8 Other Variables ---------- 15

9 Expressions and Operations ---------- 21

9-1 Arithmetic Operations 21

9-1-1 Arithmetic Symbols 21

9-1-2 Relative Value Symbols 21

9-1-3 Logic Operations 22

9-1-4 Priority of Arithmetic Operation 22

9-1-5 Data Format Conversion 23

9-2 Character String Operations 23

9-2-1 Character String Addition 23

9-2-2 Character String Comparison 24

9-3 Point Data Format 24

9-3-1 Joint Coordinate Format 24

9-3-2 Cartesian Coordinate Format 25

9-4 DI/DO Condition Expressions 25

10 Multiple Robot Control ---------- 26

10-1 Overview 26

10-2 Command Table for each Group 27

11 Command Statements ---------- 29

A B S R S T Statements 29
A C C E L Statements
(Acceleration Setting Statement for Main Group) 30
A C C E L 2 Statements
(Acceleration Setting Statement for Sub Group) 31
A R C H Statements
(Arch Position Setting Statement for Main Group) 32
A R C H 2 Statements
(Arch Position Setting Statement for Sub Group) 33
A S P E E D Statements
(Automatic Moving Speed Setting Statement for Main Group) 34
A S P E E D 2 Statements
(Automatic Moving Speed Set-ting Statement for Sub Group) 35
A X W G H T Statements
(Axis Tip Weight Setting State-ment for Main Group) 36
A X W G H T 2 Statements
(Axis Tip Weight Setting State-ment for Sub Group) 37
C A L L Statements 38
C U T Statements 40
D E C L A R E Statements 41
D E F F N Statements 43
D E L A Y Statements 44
D I M Statements (Array Variable Declaration Statement) 45
D O Statements (Output) 46
D R I V E Statements 47
D R I V E 2 Statements 49
D R I V E I Statements 51
D R I V E I 2 Statements 53
E X I T F O R Statements 55
E X I T S U B Statements 56
E X I T T A S K Statements 57
F O R and N E X T Statements 58
G O S U B and R E T U R N Statements 59
G O T O Statements 60
H A L T Statements 61
H A N D Definition Statements, C H A N G E Statements
(Hand Selection for Main Robot) 62
H A N D 2 Definition Statements, C H A N G E 2 State- ments
(Hand Selection for Sub Robot) 67
H O L D Statements 71
I F Statements 72
I N P U T Statements 74
L E T Statements (Assigning Values to Variables) 76
L O Statements (Arm lock) 80
M O Statements (Internal Output) 81

MOVE Statements 82
MOVE2 Statements 91
MOVEI Statements 96
MOVEI2 Statements 99
ON ERROR GOTO Statements 102
ON and GOTO Statements , ... 104
ON and GOSUB Statements 104
ONLINE and OFFLINE Statements 105
ORGORD Statements
(Return to Origin Sequence Setting Statement for Main Group) 106
ORGORD2 Statements
(Return to Origin Sequence Setting Statement for Sub Group) 107
ORIGIN Statements 108
OUTPOS Statements
(Out Effective Position Setting for Main Group) 109
OUTPOS2 Statements
(Out Effective Position Setting for Sub Group) 110
PDEF Statements 111
PMOVE Statements 112
PMOVE2 Statements 114
PRINT Statements 116
Pn (Point Definition Statements) 117
REM (Comments) 118
RESET Statements 119
RESTART Statements 120
RESUME Statements 121
RIGHTY and LEFTY Statements 122
RIGHTY2 and LEFTY2 Statements 123
Sn (Shift Coordinate Definition Statement) 124
SELECT CASE Statements 125
SEND Statements 127
SERVO Statements 129
SERVO2 Statements 130
SET Statements 131
SHARED Statements 132
SHIFT Statements
(Shift Coordinate Setting Statement for Main Robot) 133
SHIFT2 Statements
(Shift Coordinate Setting Statement for Sub Robot) 134
SPEED Statements (Speed Setting Statement for Main Group) 135
SPEED2 Statements (Speed Setting Statement for Sub Group) 136
START Statements 137
SUB and END SUB Statements 138
SUSPEND Statements 141
SWI Statements 142
TO Statements (Timer) 143
TOLE Statements (Tolerance Setting Statement for Main Group) ... 144

TOLE2 Statements (Tolerance Setting Statement for Sub Group) ..145
WAIT Statements 146
WEIGHT Statements
(Weight Parameter Setting State ment for Main Robot) 148
WEIGHT2 Statements
(Weight Parameter Setting Statement for Sub Robot) 149
WHILE and WEND Statements 150
Label Statements 152

12 Functions ---------- 153

12-1 Arithmetical Functions 153
12-2 Character String Functions 169
12-3 Point Functions 173

13 Multi-tasking ---------- 177

13-1 Outline 177
13-2 Task Status 177
13-3 Task Definition 178
13-4 Starting Tasks 179
13-5 Task Status Flow 179
13-6 Task Completion 180
13-7 Completion of Other Tasks 180
13-8 Tasks Suspension 180
13-9 Starting Tasks 181
13-10 Stopping Programs 181
13-11 Program List Changing 182
13-12 Program Execution Sequence 182
13-13 Common Use of Variables 183

14 Command Statement List ---------- 184

15 Robot Language Function List ---------- 191

15-1 Arithmetical Functions 191
15-2 Character String Functions 193
15-3 Point Functions 193

16 Data File Details ---------- 194

16-1 Program Files 195
16-2 Point Data Files 199
16-3 Parameter Files 202
16-4 Shift Data Files 204
16-5 Hand Data Files 207
16-6 System Files 209
16-7 Palette Definition Files 210
16-8 Variable Files 213
16-9 Array Variable Files 216
16-10 Character Strings 219
16-11 Directory Files 220
16-12 Free Memory Status 222
16-13 Point Data Use Files 223

16-14 DI Files 224
16-15 DO Files 225
16-16 MO Files 226
16-17 LO Files 227
16-18 TO Files 228
16-19 DIO Files 229
16-20 Communication Files 230
16-21 Console Input Files 230
16-22 Console Output Files 231
16-23 Machine Reference Files 232
16-24 EOF Files 233

17 User Program Examples ----------234

17-1 Basic Operation 234
17-1-1 Point Data Written Directly into Program 234
17-1-2 Using Point Numbers 234
17-1-3 Using Shift Coordinates 235
17-1-4 Palletizing 236
17-1-4-1 Utilization of the Shift Coordinates 236
17-1-4-2 Utilization of Palette Movement 237
17-1-5 DI/DO (Digital I/O) Movement 238
17-2 Application 240
17-2-1 Pick and Place Between Two Points 240
17-2-2 Palletizing 242
17-2-3 Pick and Place of Parts Stacked in Layers 244
17-2-4 Parts Inspection 1 (Multi-tasking Example) 247
17-2-5 Parts Inspection 2 (2 Robots Example) 251
17-2-6 Sealing 254

18 Sequence Program ----------255

18-1 Creating Sequence Programs 255
18-1-1 Programming Method 255
18-1-2 Compiling 256
18-2 Running Sequence Programs 258
18-2-1 Sequence Program Step Running 258
18-3 Programming the Sequencer 259
18-3-1 Assignation Statements that May be Used with the Sequencer 259
18-3-2 Input/Output Variables that May be Used with the Sequencer 259
18-3-3 Timer Definition Statements 261
18-3-4 Arithmetical Functions (Logical Operators) Used with the Sequencer 262
18-3-5 Priority of Logical Operations 262

Appendix 264

A. Reserved Word List 264

Robot Language Command and Function Index

■ *General Command*

Commands

DECLARE 41
DEF FN 43
DIM 45
EXIT FOR 55
FOR~NEXT 58
GOSUB 59
GOTO 60
HALT 61
HOLD 71
IF 72
LET 76
ON~GOSUB 104
ON~GOTO 104
REM 118
RETURN 59
SELECT CASE 125
SWI 142
WHILE~WEND 150
Label Statements 152

■ *Robot Movement*

Commands

ABSRST 29
DRIVE 47
DRIVE2 49
DRIVEI 51
DRIVEI2 53
MOVE 82
MOVE2 91
MOVEI 96
MOVEI2 99
ORIGIN 108
PMOVE 112
PMOVE2 114
SERVO 129
SERVO2 130

■ *Input/Output Control*

Commands

DELAY 44
DO 46
LO 80
MO 81
RESET 119
SET 131
TO 143
WAIT 146

Functions

DO 46
LO 80
MO 81
TO 143

■ *Screen Control*

Commands

PRINT 116
SEND~TO SCR 127

■ Key Control

Commands

INPUT 74
SEND KEY TO˜ 127

■ RS-232C Communication Port Control

Commands

SEND CMU TO˜ 127
SEND˜TO CMU 127

■ Coordinate Control

Commands

CHANGE 62
CHANGE2 67
HAND 62
HAND2 67
RIGHTY/LEFTY 122
RIGHTY2/LEFTY2 123
SHIFT 133
SHIFT2 134

■ Status Change

Commands

ACCEL 30
ACCEL2 31
ARCH 32
ARCH2 33
ASPEED 34
ASPEED2 35
AXWGHT 36
AXWGHT2 37
OFFLINE/ONLINE 105
ORGORD 106
ORGORD2 107
OUTPOS 109
OUTPOS2 110
PDEF 111
SPEED 135
SPEED2 136
TOLE 144
TOLE2 145
WEIGHT 148
WEIGHT2 149

■ Procedure

Commands

CALL 38
EXIT SUB 56
SUB˜END SUB 138
SHARED 132

■ Task Control

Commands

CUT 40
EXIT TASK 57
RESTART 120
START 137
SUSPEND 141

■ Error Control

Commands

ON ERROR GOTO 102
RESUME 121
ERR 159
ERL 159

■ Point Operations

Commands

LET 76
Pn 117

Functions

JTOXY 173
JTOXY2 173
LOCx 15
WHERE 174
WHERE2 174
XYTOJ 175
XYTOJ2 175

■ Shift Operations

Commands

LET 76
Sn 124

Functions

LOCx 15

■ Arithmetical Functions

Functions

ABS 153
ARMTYPE 155
ARMTYPE2 156
ATN 156
COS 158
DEGRAD 159
DIST 159
INT 160
LSHIFT 160
MCHREF 161
MCHREF2 161
RADDEG 164
RSHIFT 164
SIN 164
SQR 165
TAN 165

■ Referring to Parameter

Functions

ACCEL 153
ACCEL2 154
ARCH 154
ARCH2 155
AXWGHT 157
AXWGHT2 157
ORGORD 162
ORGORD2 162
OUTPOS 163
OUTPOS2 163
TOLE 166
TOLE2 167
WEIGHT 168
WEIGHT2 168

■ Character String

Functions

CHR$ 169
LEFT$ 170
LEN 160
MID$ 170
ORD 162
RIGHT$ 171
STR$ 171
VAL 167

■ Date and Time Control

Functions

DATE$ 169
TIME$ 171
TIMER 171

Robot Language Command and Function Index for Each Robot

■ Main Robot Commands

Commands

ACCEL 30
ARCH 32
ASPEED 34
AXWGHT 36
CHANGE 62
DRIVE 47
DRIVEI 51
HAND 62
MOVE 82
MOVEI 96
ORGORD 106
OUTPOS 108
PMOVE 112
RIGHTY/LEFTY 122
SERVO 129
SHIFT 133
SPEED 135
TOLE 144
WAIT ARM 146
WEIGHT 148

Functions

ACCEL 153
ARCH 154
ARMTYPE 155
AXWGHT 157
JTOXY 173
ORGORD 162
OUTPOS 163
TOLE 166
WEIGHT 168
WHERE 174
XYTOJ 175

■ Sub Robot Commands

Commands

ACCEL2 31
ARCH2 33
ASPEED2 35
AXWGHT2 37
CHANGE2 67
DRIVE2 49
DRIVEI2 53
HAND2 67
MOVE2 91
MOVEI2 99
ORGORD2 107
OUTPOS2 110
PMOVE2 114
RIGHTY2/LEFTY2 123
SERVO2 130
SHIFT2 134
SPEED2 136
TOLE2 145
WAIT ARM2 146
WEIGHT2 149

Functions

ACCEL2 154
ARCH2 155
ARMTYPE2 156
AXWGHT2 157
JTOXY2 173
ORGORD2 162
OUTPOS2 163
TOLE2 167
WEIGHT2 168
WHERE2 174
XYTOJ2 175

1 The Yamaha Robot Language

The Yamaha Robot Language is a language developed by the Yamaha Motor Company for simple and efficient programming. Commands are very similar to BASIC (Beginner's All purpose Symbolic Code) and make even complex robot movements easy to program. In this manual we will discuss the Yamaha Robot Language and give various examples of how its statements are used.

2 Characters

The Yamaha Robot Language uses the following characters:

- *Alphabetic*
  A, B, C, D, E, F, G, H, I, J, K, L, M, N, O, P, Q, R, S, T, U, V, W, X, Y, Z

- *Numerals*
  0, 1, 2, 3, 4, 5, 6, 7, 8, 9

- *Symbols*
  *( ) [ ] + - * / ^ = < > & | ~ _ % ! # $: ; , . “ ‘ @ ?*

3 Program names

The program name is a characteristic name of a program to be made in the controller and must therefore not be used with other programs. Program names can contain combinations of up to 8 alphanumeric characters and underscored characters (_).

The functions and examples of program names having a special meaning are shown below.

a) FUNCTION
b) SEQUENCE
c) _SELECT
d) COMMON

a) FUNCTION
Functions:

- *By pressing the [USER] key in "PROGRAM" mode or "MANUAL" mode, the user function can be used.*
  By using in the "PROGRAM" mode, it is possible to input commands (MOVE, GOTO, etc.) with function keys, which are often used during program editing.
  By using in "MANUAL" mode, without executing the program, DO output or MO can be output with the function keys.

Program example:

```
' FOR MANUAL MODE
*M_F1: 'DO(20)ALTERNATE
     DO(20)=¯DO(20)
*M_F2: 'DO(21)ALTERNATE
     DO(21)=¯DO(21)
           :
*M_F6: 'DO(25)MOMENTARY
     DO(25)=1
     DO(25)=0
           :
 FOR PROGRAM MODE
*P_F1: 'MOVE P,
*P_F6: 'MOVE L,
*P_F2: 'GOTO *
           :
```

Refer to Chapter 4 "Operation", in the Operation Manual for details.

b) SEQUENCE

Functions:

- *This function as distinct from the robot program, performs processing of the robot input/output (DI, DO, MO, LO, TO) in fixed cycles. The cycle is determined by the program capacity.*
  Using this function allows a simple pseudo sequencer (or emulator) to be formed in the controller.

Program example:

```
DO(20) = ¯DO(20)
DO(25) = DI(21) AND DI(22)
MO(26) = DO(26) OR DO(25)
          :
```

Refer to "17 Sequence Program" in this manual.

c) _SELECT

Functions:

- *If this program is present when the robot program is reset, then "_SELECT" is always selected. By using this function, a program can be selected by DI input and also will always return to this program when reset.*
  Differences in processing by each type of reset,
  - *When reset from the MPB screen, the system awaits a response to a query to switch the program to "_SELECT".*
  - *When reset by custom DI (Reset signal) or online command, the system switches to the "_SELECT" program.*
  - *When a LEVEL3 or LEVEL5 execution is set, the system resets when power is turned on and then switches to the "_SELECT" program.*

Program example:

- *A program is selected according to the value which is input into DI3(). When DI3() is set at 0, the system repeatedly monitors the DI input. When DI3() is set from 1 to 3, the selection moves to each program. When DI3() is set for other than the above cases, the system quits the program that is currently running.*

```
ON ERROR GOTO *ER1
*ST:
   SELECT CASE DI3()
      CASE 0
        GOTO *ST
      CASE 1
        SWI <PART1>
      CASE 2
        SWI <PART2>
      CASE 3
        SWI <PART3>
      CASE ELSE
        GOTO *FIN
   END SELECT
   GOTO *ST
*FIN:
HALT
*ER1:
   IF ERR=&H0303 THEN *NEXT_L
   ON ERROR GOTO 0
*NEXT_L:
RESUME NEXT
```

CAUTION

Refer to the Programming Manual for information on the commands utilized in the above example.

POINT

- *When a ON ERROR statement is used, the program can make a loop without ending in an error, even if the program name specified in a SWI statement is not found.*
- *An error code occurring during the program run, is input into a variable ERR. ERR=&0303 means "Program doesn't exist".*

d) COMMON

Functions:

- *Performing the same processing with two or more robot programs is usually a waste of the programming area. To cope with this, programming the same processing in the COMMON program is recommended.*

Program example:

Program name: SAMPLE1

```
DECLARE SUB *DISTANCE (A!, B!, C!)
DECLARE *AREA
X!= 2.  5
Y!= 1.  2
CALL  *DISTANCE (2. 5, 1. 2, REF  C!)
GOSUB  *AREA
PRINT   C!,  Z!
HALT
```

Program name: SAMPLE2

```
DECLARE SUB *DISTANCE (A!, B!, C!)
DECLARE *AREA
X!= 5. 5
Y!= 0. 2
CALL *DISTANCE (5.5, 0.2, REF  C!)
GOSUB *AREA
PRINT  C!, Z!
HALT
```

Program name: COMMON

```
SUB  *DISTANCE (A!, B!, C!)
    C!=SQR (A! ^2+B!^2)
END   SUB
*AREA:
     Z! = X! * Y!
HALT
```

Related commands:

- *DECLARE, GOSUB, CALL*

4 Identifiers

The groups of characters used to express labels, variables, procedures, names, etc, are referred to as "identifiers."
Identifiers are composed of 16 or less alphanumeric characters of the underscore character ("_"). If the identifier exceeds 16 characters, the characters from the 17th on are ignored and deleted.

Example: L O O P , S U B R O U T I N E , G E T _ D A T A

5 Command Statement Format

Robot language commands are always written on a single line and are arranged in the format shown below:

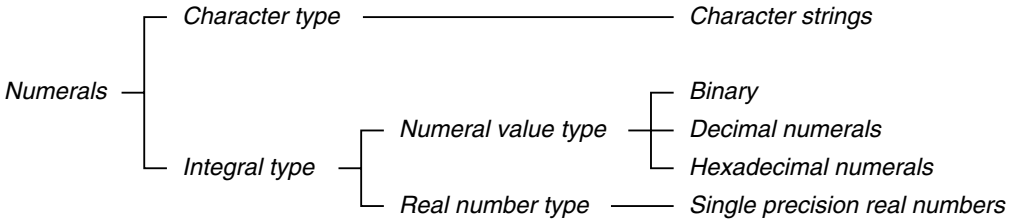
```
[<label>:]      <expression>    [<operand>]
```

POINT

- *[ ] show elements that do not have to be included in the command.*
- *< > show elements that must be written in a specific format.*
- *Elements that are not surrounded by < > are included in command as shown.*
- *| | show elements that can be interchanged with each other in the command.*
- *The label does not have to be included in the command. All labels begin with an asterisk ("*") and end with a colon (" : ").*
- *The operand can be eliminated in the case of certain commands.*
- *The commands of the program are executed in order from top to bottom unless a direction to diverge is given.*

6 Numerals

The following types of numerals are used:



6-1 Character Type Numerals

Character type numerals are delineated by double quotation marks (") and may consist of 75 bytes or less worth of characters. Strings of characters may include upper and lower case alphabetic characters, numerals, and symbols. To include a double quotation mark in a string, it is necessary to use extra double quotation marks continuously.

Examples: "YAMAHA ROBOT"
"EXAMPLE OF ""A"""
PRINT "COMPLETED"

6-2 Value Type Numerals

6-2-1 Integer Type Numerals

1. Integers

These integers from -32768 to +32767 may be used.

2. Binary

Binary numbers of 8 bits or under may be used. "&B" is used at the head of the number to define it as a binary value.

3. Hexadecimal

Hexadecimal values from 0 to FFFF may be used. "&H" is used at the head of the number to define it as a hexadecimal value.

6-2-2 Real Number Type Numerals

1. Single Precision Real Numbers

Real numbers from - 999999.9 to +999999.9 may be used (7 digits including integers and decimals). .0000001 is also possible.

2. Single Precision Real Numbers in Exponent Form

*Numbers from -1.0*1038 to +1.0*1038 may be used. Mantissas may be 7 digits long, including integers and decimals.*

Examples: - 1 . 2 3 4 5 6 E - 1 2
3 . 1 4 E 0
1 . E 5

7 Variables

Reserve words with same names as variable terms and variables starting with FN, DIn, DOn, Pn or Sn (n=0 to 9), may not be used as variables. Neither S nor P are permitted as variable names.
Variables are classified into dynamic variables and static variables. Static variables have the following names.

Integer type	*SGIn (n: 0 to 7)*
Real Number type	*SGRn(n: 0 to 7)*

Examples:
COUNT ---------- permitted
ABS ---------- not permitted
FNAME ---------- not permitted
DI1 ---------- not permitted
DO31 ---------- not permitted
P12 ---------- not permitted
S91 ---------- not permitted

7-1 Valid Range of Variables

7-1-1 Valid Range of Dynamic Variables

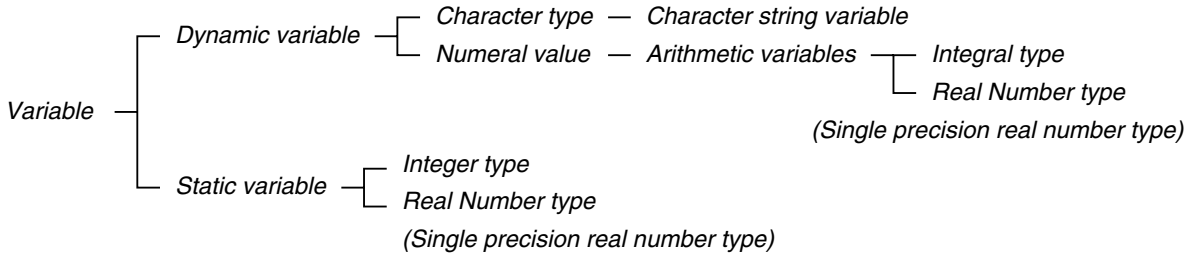
Dynamic variables are classified into dynamic global variables and dynamic local variables according to their position in the program.

- *Dynamic gloval variables are exclusive of sub-procedures. Dynamic gloval variables exist outside of program elements enclosed by SUB statements and END SUB statements.*
- *Dynamic local variables are used in sub-procedures. Dynamic local variables are only valid for use in these sub-procedures.*

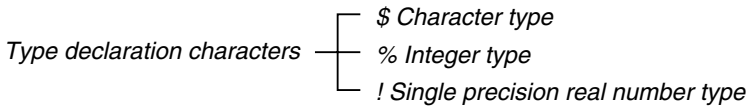
7-1-2 Valid Range of Static Variables

Static variable can always be used as global variables regardless of program statements.

7-2 Types of Variables



The type of variable is specified at the end of the variable name.



If no type declaration character is placed at the end of the variable, the variable is considered to be a simple precision real number variable (defined with "!").

Examples: C O U N T % ---------- *Integer type variable*
C O U N T ! ---------- *Simple precision real number variable*
C O U N T ---------- *Simple precision real number variable*
S T R I N G $ ---------- *Character type variable*

7-3 Array Variables

An array variable can express a series of distinct values. The elements of the array can be integers or whatever is represented by the expressions delineated by commas succeeding the variable name (see below). The length of the array is defined by the DIM (DIMension) statement (see page 36). The expressions begin with 0, but in this case, 0 represents the first value in the array. Array values may be coordinates in up to three dimensions.
All array variables are dynamic variables.

Format :

<variable name> [|%|!|$|] (<expression>, [<expression>, <expression>])

Examples: A % (1) ---------- *Integer array*
D A T A (1 , 1 0 , 3) ---- *Single precision real number array*
STRING$(10) ---------- *Character type array*

POINT

To distinguish between variables and array variables, variables are referred to as variables and array variables are referred to as array variables.

7-4 Clearing Variables

7-4-1 Clearing Dynamic Variables

In the cases below, integral type variables are cleared to zero, and character type variable are cleared to null string. The variable array is also same.

- *In the compiling PROGRAM mode, when it was routinely quit. (Refer to Chapter 4 "Operation" on the User's Manual.)*
- *After compiling a program in AUTO mode, when compiling was routinely quit.*
  (Refer to Chapter 4 "Operation" on the User's Manual.)
- *When* [F 1] *(RESET) is executed in AUTO mode.*
  (Refer to Chapter 4 "Operation" on the User's Manual.)
- *When custom input signal DI15 (program reset input) is turned on, while the program is being stopped in AUTO mode.*
  (Refer to Chapter 5 "I/O Interface" on the User's Manual.)
- *When either of following is initialized in SYSTEM mode.*
  1. *Program memory (SYSTEM>INIT>MEMORY>PROGRAM)*
  2. *Entire memory (SYSTEM>INIT>MEMORY>ALL)*
     (Refer to Chapter 4 "Operation" on the User's Manual.)
- *When SWI command is executed with* [F 7] *(DIRECT) in AUTO mode.*
  (Refer to Chapter 4 "Operation" on the User's Manual.)
- *When online command @RESET, @INIT PGM, @INIT MEM, @INIT ALL, @SWI are executed.*
  (Refer to Chapter 6 "RS-232C Interface" on the User's Manual.)
- *When the SWI statement is executed during the program.*
- *When the HALT statement is executed during the program.*

7-4-2 Clearing Static Variables

In the cases below, integer type variables and real number type variables are cleared to zero.

- *When the following is initialized in SYSTEM mode.*
  Entire memory (SYSTEM>INIT>MEMORY>ALL)
  (Refer to Chapter 4 "Operation" in the Operation Manual.)

- *When the online commands @INIT MEM, @INIT ALL are executed.*
  (Refer to Chapter 6 "RS-232C Interface" in the Operation Manual.)

8 Other Variables

1. Point data variables

Point numbers are defined with integers or expressions. The point data variable is written with a "P" followed by a value of 4 digits or less, or a expression surrounded by brackets ("[ ]").

Format :

P n n n n or P " ["<expression>"]" n = 0 to 9

(The quotation marks around the brackets do not mean that they can be eliminated.)

Examples: P 0 , P 1 1 0
P [A] , P [S T A R T P O I N T] , P [A (1 0)]

2. Shift coordinate variable

Shift numbers are defined with integers or expressions. The shift coordinate variable is written with a "S" followed by a value of 1 digit, or a expression surrounded by brackets ("[ ]").

Format :

S n or S "["<expression>"]" n = 0 to 9

(The quotation marks around the brackets do not mean that they can be eliminated.)

Examples: S 1
S [A] , S [B A S E] , S [A (1 0)]

3. Point data element variables

Point data element variables express point data by axis.

Format :

L O C x (<point expression>) x : X , Y , Z , R , A , B

Examples: A(1)= L O C X (P 1 0)
---- The X-axis data of point P10 will be the value of A(1).
L O C Z (P [A]) = 1 0 0 . 0
---- The Z-axis data of P[A] will be 100.0.

4. Shift element variables

Shift element variables express shift point data by axis.

Format :

L O C x (<shift expression>)	x : X , Y , Z , R

Examples: A(1)=L O C X (S 1)

---- The X data of S1 will be A(1).

L O C R (S [A]) = 4 5 . 0

---- The R data of S[A] will be 45.0ÅK.

5. Input Variables

Input variables express the status of the input signal.

Format 1:

D I m ([b, • • •, b])	m: Port Number 0 to 7, 10 to 13 b: Bit definition 0 to 7

If the "[b, • • •, b]" is eliminated from the expression, all eight bits are expressed.

Format 2:

D I (m b, • • •, m b)	m: Port Number 0 to 7, 10 to 13 b: Bit definition 0 to 7

POINT

Be sure to define bits in ascending order from the right.

Examples:A% = D I 1 ( )

------ The value of variable A% substitutes for the input status of ports DI(17) to DI(10).

A% = D I 5 (7 , 4 , 0)

------ The value of variable A% substitutes for the input status of DI(57), DI(54), and DI(50).

(If all above signals are 1(ON), A%=7.)

A% = D I (2 7 , 1 5 , 1 0)

------ The value of variable A% substitutes for the input status of DI(27), DI(15), and DI(10).

(If all above signals except DI(10) are 1(ON), A%=6.)

6. Output Variables

Output variables define the output signals and express the output status.

Format 1:

D O m ([b, • • •, b])	m: Port number 0 to 7, 10 to 11 b: Bit definition 0 to 7

If the "[b, •••, b]" is eliminated from the expression, all eight bits are expressed.

Format 2:

D O (m b, • • •, m b)	m: Port number 0 to 7, 10 to 11 b: Bit definition 0 to 7

POINT

Be sure to define bits in ascending order from the right.

Examples:A% = D O 2 ( )

------ The value of variable A% substitutes for the output status of DO(27) to DO(20).

A% = D O 5 (7 , 4 , 0)

------ The value of variable A% substitutes for the output status of DO(57), DO(54), and DO(50).

(If all above signals are 1(ON), A%=7.)

A% = D O (3 7 , 2 5 , 2 0)

------ The value of variable A% substitutes for the output status of DO(37), DO(25), and DO(20).

(If all above signals except DO(20) are 1(ON), A%=6.)

7. Internal Output Variables

Internal output variables are used to communicate between the program and a sequencer.

It is possible to change and display their content.

The internal output variables for ports 0 and 1 are special and can only be displayed.

1) *Port 0 is for the status of the origin sensors for axes 1 to 8 (in order from bit 0). "1" is ON, and "0" is OFF.*
2) *Port 1 is for hold status of axes 1 to 8 (in order from bit 0). "1" is hold, "0" is nonhold.*
   HOLD is the status for times when shifted with the MOVE command and placed within the tolerance for the target position.
   When the servo is set to OFF, the status is nonhold. The non-use axis sets to "1".

Format 1:

M O m ([b, • • •, b])	*m: Port number 0 to 7, 10 to 13* *b: Bit definition 0 to 7*

If [b, • • •, b] is eliminated, all 8 bits are expressed.

Format 2:

M O (m b, • • •, m b)	*m: Port number 0 to 7, 10 to 13* *b: Bit definition 0 to 7*

POINT

Be sure to define bits in ascending order from the right.

Examples:A = M O 2 ( )

------ The value of variable A substitutes for the internal output status of MO(27) to MO(20).

A = M O 5 (7 , 4 , 0)

------ The value of variable A substitutes for the internal output status of MO(57), MO(54), and MO(50).

(If all above signals are ON, A=7.)

A = M O (3 7 , 2 5 , 2 0)

------ The value of A substitutes for the internal output status of MO(37), MO(25), and MO(20).

8. Arm Lock Output Variables

Arm lock output variables are used to prohibit an axis movement.
It is possible to output and display a variable.
There is only 1 port and the bits starting from 0, correspond in order, to axis 1 to axis 8.
Movement of the axis which corresponds to the variable is prohibited when the variable is ON.

Format 1:

L O m ([b, • • •, b])	*m: Port number 0* *b: Bit definition 0 to 7*

If [b, • • •, b] is eliminated, all 8 bits are expressed.

Format 2:

L O (m b, • • •, m b)	*m: Port number 0* *b: Bit definition 0 to 7*

POINT

Be sure to define bits in ascending order from the right.

Examples:A% = L O 0 ( )

------ The value of variable A% substitutes for the arm lock status of LO(07) to LO(00).

A% = L O 0 (7 , 4 , 0)
------ The value of variable A% substitutes for the arm lock status of LO(07), LO(04) and LO(00).
(If all above signals are 1 (ON), A%=7.)

A% = L O (0 6 , 0 4 , 0 1)
------ The value of variable A% substitutes for the arm lock status of LO(06), LO(04) and LO(01).
(If all above signals except LO(01) are 1 (ON), A%=6.)

9. Timer Output Variables

Timer output variables are used in the timer function of a sequence program. It is possible to change and display their contents.
Timer function is valid only in the sequence program. If this variable is output in a normal program, it is an internal output like the MO variable.

Format 1:

T O m ([b, • • •, b])	*m: Port number 0* *b: Bit definition 0 to 7*

If [b, • • •, b] is eliminated, all 8 bits are expressed.

Format 2:

T O (m b, • • •, m b)	*m: Port number 0* *b: Bit definition 0 to 7*

POINT

Be sure to define bits in ascending order from the right.

Examples:A% = T O 0 ( )

------ The value of variable A% substitutes for the status of TO(07) to TO(00).

A% = T O 0 (7 , 4 , 0)

------ The value of variable A% substitutes for the status of TO(07), TO(04) and TO(00).

(If all above signals are 1 (ON), A%=7.)

A% = T O (0 6 , 0 4 , 0 1)

------ The value of variable A% substitutes for the status of TO(06), TO(04) and TO(01).

(If all above signals except TO(01) are 1 (ON), A%=6.)

9 Expressions and Operations

9-1 Arithmetic Operations

9-1-1 Arithmetic Symbols

^	Exponent operation
-	Negative
*, /	Multiplication and division
+, -	Addition and subtraction
MOD	Remainder

When the value used in remainder calculations is a real number, it is converted into integers (all decimals are ignored), and the program continues with the resulting value. The resulting value is the remainder of a division operation.

```
Examples: A = 1 5  M O D  2
                ------ A=1. (15/2=7....1)

          A = 1 7 . 3 4  M O D  5 . 9 8
                ------ A=2. (17/5=3....2)
```

9-1-2 Relative Value Symbols

=	Equal to
<>, ><	Not equal to
<	Less than
>	More than
<=, =<	Less than or equal to
>=, =>	More than or equal to

Relative value symbols are used to compare 2 values. If the result is true, a "-1" is generated. If it is false, a "0" is generated.

```
Example:  A =  1 0  >  5
                ------ 10>5 is true so A=-1
```

9-1-3 Logic Operations

NOT, ~
AND, &
OR, |
XOR

Logic operations are used to manipulate 1 or 2 values bit by bit. For example, the status of an I/O port can be manipulated. Depending on the logic operation performed, the results generated are either "0" or "1". Logic operations with real numbers convert the values into integers before they are executed.

Examples: A% = N O T 1 3 . 0 5
---------- Each bit of 13 (&B0000000000001101) is reversed, and A% becomes the result: -14 (&B11111111 11110010=&HFFF2).

A% = 3 A N D 1 0
---------- The bits that are identical (when both are "1") in 3 (&B0000000000000011) and 10 (&B00000000 00001010) are accumulated and the result A% becomes 2 (&B0000000000000010).

A% = 3 O R 1 0
---------- The "1" bits of 3 (&B0000000000000011) and 10 (&B0000000000001010) are accumulated to generate the value of A%, which becomes: 11 (&B0000000000001011).

A% = 3 X O R 1 0
---------- A "1" bit is generated when the bits of 3 (&B0000000000000011) and 10 (&B00000000 00001010) are different. A% becomes: 9 (&B0000000000001001).

9-1-4 Priority of Arithmetic Operation

1. *Expressions included in parentheses*
2. *Functions, variables*
3. *Exponents ("^")*
4. *Independent "+" and "-" signs (unary operator)*
5. *Multiplication, division*
6. *MOD*

7. *Addition, subtraction*
8. *Relative value symbols (“<“, etc.)*
9. *NOT, ~ operations*
9. *AND, & operations*
10. *OR, |, XOR operations*

Operations are performed in the above order. When two operations of equal priority appear in a statement, the operations are executed in order from left to right.

9-1-5 Data Format Conversion

Data format is converted in cases where two values of different format are involved in the same operation.

1) *When real numbers are assigned to an integer, all decimals are eliminated.*
   Example: A % = 1 2 5 . 6 7---------- A % = 1 2 5
2) *When integers and real numbers are involved in the same operation, the result becomes a real number.*
   *Example: A (0) = 1 2 5 * 0 . 2 5----- A (0) = 3 1 . 2 5*
3) *When integers are divided by integers, the result is an integer.*
   Example: A (0) = 1 0 0 / 3---------- A (0) = 3 3

9-2 Character String Operations

9-2-1 Character String Addition

Character strings may be combined by using the “+” sign.

```
Examples: A $ = “ Y A M A H A “
          B $ = “ R O B O T “
          C $ = “ L A N G U A G E “
          D $ = “ M O U N T E R “
          E $ = A $ + “   “ + B $ + “   “ + C $
          F $ = A $ + “   “ + D $
          P R I N T  E $
          P R I N T  F $

Results in: Y A M A H A  R O B O T  L A N G U A G E
            Y A M A H A  M O U N T E R
```

9-2-2 Character String Comparison

Characters can be compared with the same relative value symbols used for other values. In the case of character strings, the comparison is performed from the beginning of each string, character by character. If all characters match in both the strings, they are considered to be equal. Even if only one character in the strings differs with its corresponding character in the other string, The string with the character with the greater character code value becomes the larger string. If one string is shorter than the other, it is judged to be the string of a lesser value.

All examples below are true.

Examples: " A A " < " A B "
" X & " > " X # "
" D E S K " < " D E S K S "

Character string comparison can be used to find out the contents of character string variables, and to sort character strings in alphabetic order.

9-3 Point Data Format

There are two types of point data formats: joint coordinate format and Cartesian coordinate format.
Point numbers are in the range of 0 to 4000.
(This range is limited to 0 to 1600 for the MRC series with no extension RAM.)

9-3-1 Joint Coordinate Format

±nnnnnn (same for X, Y, Z, R, A, B axes)
6 digits or less, decimal numbers with plus or minus sign.
Unit: pluses

9-3-2 Cartesian Coordinate Format

±nnn.nn to ±nnnnn. (same for X, Y, Z, R, A, B axes)
2 decimal places or less, decimal numbers with 7 digits or less, with plus or minus sign.

Unit: *X* *[mm], [deg]*
Y *[mm], [deg]*
Z *[mm], [deg]*
R *[mm], [deg]*
A *[mm], [deg]*
B *[mm], [deg]*

CAUTION
"+" sign may be eliminated.

9-4 DI/DO Condition Expressions

DI/DO condition expressions may be used to set conditions for MOVE STOPON (see P.73) and WAIT statements (see P.124). The integers, variables, and arithmetic symbols that may be used with DI/DO condition expression are shown below.

a. *Integers*
   Decimal integers, binary integers, hexadecimal integers
b. *Variables*
   Global integer type, global real number type
c. *Arithmetic Symbols*
   Relative value symbols, logic operations

d. *Operation Priority*
   1. *Relative value symbols*
   2. *NOT, ~*
   3. *AND, &*
   4. *OR, |, XOR*

   Example: WAIT DI(31)=1 OR DI(34)=1
   ------ Robot will wait until either DI(31) or DI(34) are ON.

10 Multiple Robot Control

10-1 Overview

The robot controller can control multiple robots.
In addition, using the multi-tasking function (refer to Section 12 "Multi-tasking") enables multiple robots to move asynchronously. To use this function, settings for two robots or settings for auxiliary axes must be made in the system generation at the time of shipment.

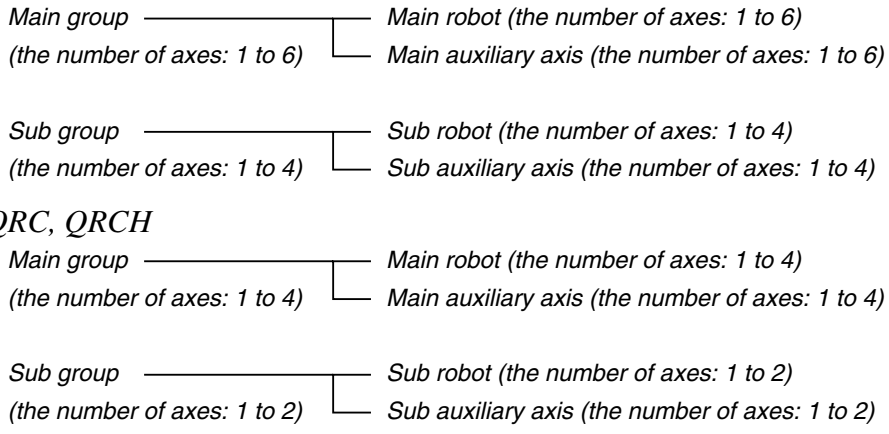
A robot axis is classified into one of the groups below.

MRC,MRCH
- *Main group (6 axes)*
- *Main group (2 axes) + sub group (2 axes)*
- *Main group (4 axes) + sub group (4 axes)*
- *Main group (6 axes) + sub group (2 axes)*

QRC, QRCH
- *Main group (4 axes)*
- *Main group (2 axes) + sub group (2 axes)*

A main group is composed of a main robot and main auxiliary axes, and a sub group is composed of a sub robot and sub auxiliary axes.
Settings are only for main group of main robot when setting one robot without auxiliary axis.
When no settings have been made for main auxiliary axes or for sub auxiliary axes, then the main group is composed only of the main robot, and the sub group is composed only of the sub robot.
The number of axes on a main robot or a sub robot is 0 only when using a MULTI type robot and all the axes are set as the auxiliary axes. (all single axis specifications)

MRC,MRCH


Main group
(the number of axes: 1 to 6)
Main robot (the number of axes: 1 to 6)
Main auxiliary axis (the number of axes: 1 to 6)
Sub group
(the number of axes: 1 to 4)
Sub robot (the number of axes: 1 to 4)
Sub auxiliary axis (the number of axes: 1 to 4)

QRC, QRCH

Main group
(the number of axes: 1 to 4)
Main robot (the number of axes: 1 to 4)
Main auxiliary axis (the number of axes: 1 to 4)
Sub group
(the number of axes: 1 to 2)
Sub robot (the number of axes: 1 to 2)
Sub auxiliary axis (the number of axes: 1 to 2)

10-2 Command Table for each Group

The special commands and functions for robot movement and coordinates control are shown following.

Classification	Main Group	Sub Group
Robot Movement	DRIVE, DRIVEI, MOVE, MOVEI, PMOVE, SERVO, WAIT ARM	DRIVE2, DRIVEI2, MOVE2, MOVEI2, PMOVE2, SERVO2, WAIT ARM2
Coordinates Control	CHANGE, HAND, LEFTY/RIGHTY, SHIFT	CHANGE2, HAND2, LEFTY2/RIGHTY2, SHIFT2
Status Change	ACCEL, ARCH, ASPEED, AXWGHT, ORGORD, OUTPOS, SPEED, TOLE, WEIGHT	ACCEL2, ARCH2, ASPEED2, AXWGHT2, ORGORD2, OUTPOS2, SPEED2, TOLE2, WEIGHT2
Point Operation	JTOXY, WHERE, XYTOJ	JTOXY2, WHERE2, XYTOJ2
Parameter Change	ACCEL, ARCH, AXWGHT, ORGORD, OUTPOS, TOLE, WEIGHT	ACCEL2, ARCH2, AXWGHT2, ORGORD2, OUTPOS2, TOLE2, WEIGHT2

CAUTION

1. ***MOVE (MOVE2) and MOVEI (MOVEI2) commands are used to move a main robot (a sub robot).***

   But the axis which is set as an auxiliary axis cannot be moved with MOVE (MOVE2), MOVEI (MOVEI2) or PMOVE (PMOVE2) command. Use DRIVE (DRIVE2) or DRIVEI (DRIVEI2) command to move it.

   Example:
   Main group ..... Main robot (2 axes) + Auxiliary axis (2 axes)
   Sub group..... Sub robot (2 axes) + Auxiliary axis (2 axes)

   When a robot is composed as in the above example, MOVE (MOVE2) and MOVEI (MOVEI2) commands can move only the main robot (the sub robot).
   Use DRIVE (DRIVE2) or DRIVEI (DRIVEI2) command to move an auxiliary axis in the main group (the sub group).

2. ***Linear interpolation or circular interpolation using the MOVE statement are only possible with task 1 (main task) and direct command.***
3. ***PTP control is possible for the MOVE2 statement. Linear interpolation and circular interpolation are inoperable.***

CAUTION

4. ***When specifying all the axes with SERVO or SERVO2 command, the servos of all the axes in the main group and the sub group can be changed ON/OFF.***
5. ***The hands which can be used with CHANGE (CHANGE2) or HAND (HAND2) command are H0 to H3 (H4 to H7).***
6. ***The SPEED (SPEED2) command and WHERE (WHERE2) function are executed for all the axes in each group.***
   Similarly, when specifying all the axes with ACCEL (ACCEL2) command, the acceleration coefficients for all the axes in the main group (the sub group) can be changed.
7. ***The WEIGHT (WEIGHT2) command is used to change the value of tip weight parameter for the main robot (the sub robot). This command does not effect any main auxiliary axes (any sub auxiliary axes) at all. Use AXWGHT (AXWGHT2) command to change the value of axis tip weight of the main auxiliary axis (the sub auxiliary axis).***

11 Command Statements

A B S R S T Statements

FORMAT:

```
A B S R S T
```

EXPLANATION:
This command statement executes absolute motor axis origin return for the robot. A shutdown while movement is in-progress, will cause an incomplete return to origin.
When setting two robots, return to origin movement for the sub robot group is performed after completing return to origin for main group, and then absolute reset is executed. After executing this command, the power supply must be turned on again.

EXAMPLE :
A B S R S T---------- Performs absolute motor return to origin.

RELATED COMMAND: ORIGIN, ORGORD, ORGORD2, MCHREF, MCHREF2

A C C E L Statements (Acceleration Setting Statement for Main Group)

FORMAT 1:

```
A C C E L      <expression>
```

FORMAT 2:

```
A C C E L (<expression 1>)=<expression 2>
```

The value of <expression 1> must be from 1 to 6 (axis number).

EXPLANATION:
This command changes the acceleration coefficient of the acceleration parameter for the main group to the value defined in the <expression>.
Format 1 changes all the axes in the main group. Format 2 changes the coefficient of acceleration of the axis specified in <expression 1> to the value in <expression 2>.

POINT

- *Axis acceleration parameters for robot configuration axes and auxiliary axes are changed.*
- *If an axis is set when "no axis" has been specified in "GENERATE" mode, there will be an error message "Specification mismatch" to remind the user of the conflict in usage. The execution of the program will also be halted.*

EXAMPLE :

```
A=50
ACCEL A
ACCEL(3)=100
'CYCLE WITH INCREASING ACCELERATION
FOR A=10 TO 100 STEP 10
ACCEL A
    MOVE P, P0
    MOVE P, P1
NEXT A
HALT "END TEST"
```

A C C E L 2 Statements (Acceleration Setting Statement for Sub Group)

FORMAT 1:

```
A C C E L 2    <expression>
```

FORMAT 2:

```
A C C E L 2    (<expression 1>)=<expression 2>
```

The value of <expression 1> must be from 1 to 4 (axis number).

EXPLANATION:
This command changes the acceleration coefficient of the acceleration parameter for the sub group to the value defined in the <expression>.
Format 1 changes all the axes in the sub group. Format 2 changes the coefficient of acceleration of the axis specified in <expression 1> to the value in <expression 2>.

POINT

- *This command is valid only when the sub group has been set in system generation.*
- *If an axis is set when "no axis" has been specified in system generation, there will be an error message "Specification mismatch" to remind the user of the conflict in usage. The execution of the program will also be halted.*

E X A M P L E :

```
A=50
ACCEL2 A
ACCEL2(3)=100
'CYCLE WITH INCREASING ACCELERATION
FOR A=10 TO 100 STEP 10
    ACCEL2 A
    MOVE2 P, P0
    MOVE2 P, P1
NEXT A
HALT "END TEST"
```

A R C H Statements (Arch Position Setting Statement for Main Group)

FORMAT 1:

```
A R C H <expression>
```

FORMAT 2:

```
A R C H (<expression 1>)=<expression 2>
```

The value of <expression 1> must be from 1 to 6 (axis number).

EXPLANATION:
This command statement changes the arch position parameter for the main group to the value specified in <expression>. Format 1 changes all axes of the main group. Format 2 changes the arch position parameter for the axis specified in <expression 1> to the value specified in <expression 2>.

POINT

If an axis is set when "no axis" has been specified in "GENERATE" mode, there will be an error message "Specification mismatch" to remind the user of the conflict in usage. The execution of the program will also be halted.

EXAMPLE:

```
'CYCLE WITH INCREASING ARCH POSITION
DIM SAV(3)
GOSUB *SAVE_ARCH
FORA=1000 TO 10000 STEP 1000
    GOSUB *CHANGE_ARCH
    MOVE P,P0,Z=0
    DO3(0)=1---------------- Chuck closes
    MOVE P,P1,Z=0
    DO3(0)=0---------------- Chuck opens
NEXT A
GOSUB *RESTORE_ARCH
HALT
*CHANGE_ARCH:
    FOR B=1 TO 4
      ARCH(B)=A
    NEXT B
RETURN
*SAVE_ARCH:
    FOR B=1 TO 4
      SAV(B-1)=ARCH(B)
    NEXT B
RETURN
*RESTORE_ARCH:
    FOR B=1 TO 4
      ARCH(B)=SAV(B-1)
    NEXT B
RETURN
```

A R C H 2 Statements (Arch Position Setting Statement for Sub Group)

FORMAT 1:

```
A R C H 2 <expression>
```

FORMAT 2:

```
A R C H 2 (<expression 1>)=<expression 2>
```

The value of <expression 1> must be from 1 to 4 (axis number).

EXPLANATION:
This command statement changes the arch position parameter for the sub group to the value specified in <expression>. Format 1 changes all axes of the sub group. Format 2 changes the arch position parameter for the axis specified in <expression 1> to the value specified in <expression 2>.

POINT

- *This command is valid only when the sub group has been set in system generation.*
- *If an axis is set when "no axis" has been specified in system generation, there will be an error message "Specification mismatch" to remind the user of the conflict in usage. The execution of the program will also be halted.*

EXAMPLE:

```
'CYCLE WITH INCREASING ARCH POSITION
DIM SAV(3)
GOSUB *SAVE_ARCH
FOR A=1000 TO 10000 STEP 1000
    GOSUB *CHANGE_ARCH
    MOVE2 P,P0,Z=0
    DO3(0)=1---------- Chuck closes
    MOVE2 P,P1,Z=0
    DO3(0)=0---------- Chuck opens
NEXT A
GOSUB *RESTORE_ARCH
HALT
*CHANGE_ARCH:
    FOR B=1 TO 4
      ARCH2(B)=A
    NEXT B
RETURN
*SAVE_ARCH:
    FOR B=1 TO 4
      SAV(B-1)=ARCH2(B)
    NEXT B
RETURN
*RESTORE_ARCH:
    FOR B=1 TO 4
      ARCH2(B)=SAV(B-1)
    NEXT B
RETURN
```

A S P E E D Statements (Automatic Moving Speed Setting Statement for Main Group)

FORMAT:

```
A S P E E D <expression>
```

The value of <expression> must be from 1 to 100. (Unit: %)

EXPLANATION:
This command changes the automatic moving speed for the main group to the value defined in the <expression>.

POINT

- *Changes all the speeds for robot configuration axes and auxiliary axes.*
- *The operating speed is set to the product of the automatic moving speed which is set by MPB operation or ASPEED commands and the speed which is specified by the SPEED command in the program.*

  Example: When the automatic moving speed is 80% and the speed set by the SPEED command is set to 50% then:
  *Moving speed =80%*50%=40%.*

EXAMPLE:

```
S P E E D  7 0
A S P E E D  1 0 0
M O V E  P ,  P 0
        ----------------- Move at 70%(=100*70) of speed from current position to P0.
A S P E E D  5 0
M O V E  P ,  P 1
        ----------------- Move at 35%(=50*70) of speed from current position to P1.
M O V E  P ,  P 2 ,  S = 1 0
        ----------------- Move at 5%(=50*10) of speed from current position to P1.
```

RELATED COMMAND: ASPEED2, SPEED, SPEED2

A S P E E D 2 Statements (Automatic Moving Speed Set-ting Statement for Sub Group)

FORMAT:

```
A S P E E D 2 <expression>
```

The value of <expression> must be from 1 to 100. (Unit: %)

EXPLANATION:
This command changes the automatic moving speed for the sub group to the value defined in the <expression>.

POINT

- *Changes all the speeds for robot configuration axes and auxiliary axes.*
- *The operating speed is set to the product of the automatic moving speed which is set by MPB operation or ASPEED2 commands and the speed which is specified by the SPEED2 command in the program.*

  Example: When the automatic moving speed is 80% and the speed set by the SPEED2 command is set to 50% then:
  *Moving speed =80%*50%=40%.*

EXAMPLE:

```
S P E E D 2  7 0
A S P E E D 2  1 0 0
M O V E 2  P ,  P 0
        ---------- Move at 70%(=100*70) of speed from current position to P0.
A S P E E D 2  5 0
M O V E 2  P ,  P 1
        ---------- Move at 35%(=50*70) of speed from current position to P1.
M O V E 2  P ,  P 2 ,  S = 1 0
        ---------- Move at 5%(=50*10) of speed from current position to P1.
```

CAUTION

This command is valid only when the sub group has been set in system generation.

RELATED COMMAND: ASPEED, SPEED, SPEED2

A X W G H T Statements (Axis Tip Weight Setting State-ment for Main Group)

FORMAT:

A X W G H T (<expression 1>)=<expression 2>

The value of <expression 1> must be from 1 to 6 (axis number).

EXPLANATION:
This command changes the axis tip weight parameter for the axis (in the main group) specified in <expression 1> to the value specified in <expression 2>.

POINT

- *Axis tip weight of a specified axis is changed.*
- *It is possible to change the axis tip weight only when the auxiliary axis or the robot type is MULTI.*
- *This command is valid only when the main robot is a MULTI type robot or executing to the main auxiliary axis.*
- *Robot type and the auxiliary axes are set at the time of shipment.*

EXAMPLE:

```
A = 5
B = 0
C = A X W G H T ( 1 ) ---------- Evacuation
A X W G H T ( 1 ) = A
D R I V E  ( 1 , P 0 )
A X W G H T ( 1 ) = B
D R I V E  ( 1 , P 1 )
A X W G H T ( 1 ) = C ---------- Restoration
H A L T
```

A X W G H T 2 Statements (Axis Tip Weight Setting State-ment for Sub Group)

FORMAT:

```
A X W G H T 2 (<expression 1>)=<expression 2>
```

The value of <expression 1> must be from 1 to 4 (axis number).

EXPLANATION:
This command changes the axis tip weight parameter for the axis (in the sub group) specified in <expression 1> to the value specified in <expression 2>.

POINT

This command is valid only when the sub robot is a MULTI type robot or executing to the sub auxiliary axis.
Robot type and the auxiliary axes are set at the time of shipment.

EXAMPLE:

```
A = 5
B = 0
C = A X W G H T 2 ( 1 ) -------- Evacuation
A X W G H T 2 ( 1 ) = A
D R I V E 2  ( 1 , P 0 )
A X W G H T 2 ( 1 ) = B
D R I V E 2  ( 1 , P 1 )
A X W G H T 2 ( 1 ) = C -------- Restoration
H A L T
```

C A L L Statements

FORMAT:

```
C A L L <label>[(<parameter>[, <parameter>, ...])]
```

EXPLANATION:
This statement calls sub-procedures defined by the SUB, END SUB statements.
The <label> is the name of the sub-procedure defined with a SUB statement. The <parameter> expressions supply the necessary data for the sub-procedure to be executed.

The following parameters may be passed on to a sub-procedure that is called.

- *Numerals (numeral value or character) and expressions*
  When numerals or expressions are passed on to the subroutine, it is the value of the expression or variable that is passed on.
- *Variables*
  Examples

1	*Simple variables*	*A%, REF B!, C$*
2	*The entire array*	*A!(), B$()*
3	*Specific element of an array*	*A(1, 2), REF B%(2), C$(10)*

POINT

When simple variables or specific elements of an array are used, the value of the variable is passed on to the subroutine. Even if the value changes within the sub-procedure, the value of the original variable will not change.
To pass on the variable itself or specific element of an array itself to the subroutine, and not just the value, add the "REF" statement. In this case, if the value of the variable or array element changes during use in the subroutine, it will continue at its new value when the program is returned to.
If an entire array is to be passed on, use the "REF" statement.

EXAMPLE:

```
X% = 4
Y% = 5
CALL *COMPARE(REF X%, REF Y%)
HALT
'
SUB *COMPARE(A%, B%)
    IF A%<B% THEN
       TEMP% = A%
       A% = B%
       B% = TEMP%
    ENDIF
END SUB
```

RELATED COMMAND: DECLARE, EXIT SUB, SUB, END SUB

POINT

To end a subprocedure which has been called with the CALL statement, always use the END SUB statement or EXIT SUB statement.
If another statement such as GOTO is used to exit from the subprocedure, an error such as "Stack Overflow" may occur.

CUT Statements

FORMAT:

```
CUT Tn
```

n = 2 to 8

EXPLANATION:
This command forces the ending of the execution of a separate task that is under execution or temporarily suspended.

POINT

- *To use this command, a high-speed arithmetic processor (option) must be provided.*
- *This command cannot be used to have a routine end itself.*

EXAMPLE:

```
        START *SUBTASK, T2
        A=0
*L0:
        A=A+1
        IF A=11 THEN CUT T2
           ------------------------------ When A=11, SUBTASK is ended.
        MOVE P, P0
        MOVE P, P1
        GOTO *L0
*SUBTASK:
        DO2(0)=1
        DELAY 1000
        DO2(0)=0
        GOTO *SUBTASK
```

RELATED COMMAND: EXIT TASK, RESTART, START, SUSPEND

D E C L A R E Statements

FORMAT 1:

```
D E C L A R E <label>[, <label>...]
```

<label>: A label defined by an external program.

FORMAT 2:

```
D E C L A R E  S U B <label>[(<parameter>[, <parameter>]...)]
```

<label> : A label defined by an external program.
<parameter> : Parameters for sub-procedures. The parameters specified here are the number of parameters needed for the sub-procedure and the format of the data.

EXPLANATION:
This command statement declares that a label or sub-procedure are in an external program. In the case of a sub-procedure, the format of the data is checked.
The DECLARE statement can be used in programs other than the COMMON program (it may not be used in a sub-procedure). It is effective for the duration of the program.

POINT

- *External programs can only use COMMON.*
- *Only GOSUB, CALL, ON and GOSUB statement may use external labels.*

EXAMPLES:
Common labels in external program.
Program name: DIST1

```
'=====================================
'    M A I N   P R O G R A M
'=====================================
D E C L A R E  *D I S T A N C E ,  *A R E A
X ! = 2 . 5
Y ! = 1 . 2
G O S U B  *D I S T A N C E
G O S U B  *A R E A
H A L T
```

Program name: COMMON

```
'=====================================
'    'COMMON' PROGRAM
'=====================================
*DISTANCE:
  PRINT X!^2+Y!^2
  RETURN
*AREA:
  PRINT X!*Y!
  RETURN
```

Common sub-procedures in external program
Program name: DIST2

```
'=====================================
     MAIN PROGRAM
'=====================================
DECLARE SUB *DISTANCE(X!, Y!, D!)
DECLARE SUB *AREA(X!, Y!, A!)
CALL *DISTANCE(2.5, 1.2, REF D!)
PRINT D!
CALL *AREA(2.5, 1.2, REF A!)
PRINT A!
HALT
```

Program name: COMMON

```
'=====================================
'    'COMMON' PROGRAM
'=====================================
SUB *DISTANCE(X!, Y!, D!)
D!=X!^2+Y!^2
END SUB
SUB *AREA(X!, Y!, A!)
A!=X!*Y!
END SUB
```

RELATED COMMAND: CALL, EXIT SUB, GOSUB, ON~GOSUB, SUB, END SUB

D E F F N Statements

FORMAT:

D E F F N <name> [	*%*	*]*	*[(<parameter>, <parameter>• • • • •)]*
	!		*=<function definition statement>*
	$		

EXPLANATION:
DEF FN is a command that the user may use to define a function. The function that is defined may be called in the following format: FN name (variable).
The name is 16 characters long or less, including the characters: "FN".
The <parameters> are the variable names of the variables used in the <function definition statement>. These variables are effective only when the <function definition statement> is evaluated. There may be other variables with the same name in the program.
When the function that uses the parameters is called, the parameters' format (numeral, variable, or expression) is the same.

POINT

- *You may use numeral value variables or character type variables in the <parameter>.*
- *If a variable in the <function definition statement> has not been included in the list of variable names (<parameter>), the value assigned to that particular variable name is used for the calculation.*
- *Be sure to put a space between "DEF" and "FN", otherwise, the result will be considered a variable.*

EXAMPLE:

```
D E F  F N P A I = 3 . 1 4 1 5 9 2
D E F  F N A S I N ( X ) = A T N ( X / S Q R ( - X ^ 2 + 1 ) )
    •
    •
A = F N A S I N ( B ) * 1 0
```

D E L A Y Statements

FORMAT:

D E L A Y <expression>

The value of <expression> must be from 10 to 655340. (Unit: ms)

EXPLANATION:
The DELAY statement will cause the robot to delay movement for the period of time defined by the <expression>. The delay time is set in milliseconds, and the lowest allowable value is 10 milliseconds.

EXAMPLE:

```
D E L A Y  3 5 0 0---------- Robot waits for 3500ms (3.5 seconds).
D E L A Y  A * 1 0
```

POINT

◯ *The MRC controllers from V5.06 onward, do not pause after positioning.*

```
MOVE P, P1
DELAY 1000
```

If you make a setting as in the above example, the robot will delay movement for 1 second from the time of entering the out effective position.

```
MOVE L, P1
DELAY 1000
```

If you make a setting as in the above example, the robot will delay movement for 1 second just from the time of starting to move to P1.

D I M Statements (Array Variable Declaration Statement)

FORMAT:

```
D I M <array definition> [, <array definition>,.....]
```

ARRAY DEFINITION:

```
<name> [ | % | ] (<numeral> [, <numeral> [, <numeral>] ])
         | ! |
         | $ |
```

EXPLANATION:
This statement declares the name and length (dimension) of an array variable. A maximum of 3 dimensions may be used for the array. Several arrays can be declared in 1 line by differentiating them with a "," (comma).
By punctuating with a "," (comma) several arrays can be stated in 1 line.

EXAMPLES:

```
D I M  A % ( 1 0 )
       ----------------- Defines an array of 11 integers (A%(0) to A%(10)).

D I M  B ( 2 ,  3 ,  4 )
       ----------------- Defines a matrix of three dimensions and 60 real
                         number values from B(0, 0, 0) to B(2, 3, 4).

D I M  C % ( 2, 2 ), D ! ( 1 0 )
       ----------------- Defines an integer array from C%(0,0) to C%(2,2)
                         and a real number array from D!(0) to D!(10).
```

POINT

Each array or dimension of a matrix is labelled from 0, and therefore consists of a number of entries equal to the DIM value + 1.

D O Statements (Output)

FORMAT:

[L E T]	*D O m ([b, • • • , b])* *D O (m b , • • • , m b)*	*=<expression>*

m: Port number 2 to 7, 10 to 11
b: Bit definition 0 to 7

EXPLANATION:
This command statement outputs the specified value to a DO (digital output) port. When the arm is in motion, the specified value is output at the point that movement has been completed (in other words, the arm reaches the OUT effective position). However, DO(27) cannot be used for the QRCH-E controller as it is already used as a custom output.

POINT

Output is not possible to the DO0 and DO1 ports.
Bit should be specified in ascending order from the right.

EXAMPLES:

D O 2 ( ) = & B 1 0 1 1 1 0 0 0
---------- DO(23), (24), (25), (27) are ON, and DO(20), (21), (22), (26) are OFF.

D O 2 (6 , 5 , 1) = & B 0 1 0
---------- DO(25) is ON, and DO(21), (26) are OFF.

D O 2 ( ) = 1 5
---------- DO(20), (21), (22), (23) are ON, and DO(24), (25), (26), (27) are OFF.

D O (3 7 , 3 5 , 2 7 , 2 0) = A
---------- The 4 lower bits of the variable A is output to DO(37), (35), (27), (20).

RELATED COMMAND: RESET, SET

D R I V E Statements

FORMAT:

```
DRIVE (<expression 1>, | <expression 2> )      |[,(<expression 1>, | <expression 2> )      |...]
                       | <point expression>>   |                  | <point expression>>  |
                                                [,| SPEED          | =<expression 3>][,XY]
                                                  | S            |
```

EXPLANATION:

The DRIVE statement executes an absolute movement command for specified axes in the main group.

Axis movement begins after positioning the arms of all axes specified in <expression 1> (to within the tolerance range), and the command terminates when the arms enter the OUT effective position. (Refer to "12-1-2 Axis Parameter" in the Controller User's Manual.)

Use the WAIT ARM statement (described later) for continuing the subsequent task after positioning to within the tolerance range.

Specify the axis number in <expression 1> and the position in <expression 2> or <point expression>. When <point expression> is used to specify the position, the data for the axis specified in <expression 1> is used. At this point, if the position data is in "mm/deg" units (real type variable or constant containing a decimal point), the linear axes from the "0" pulse point move to the positions specified in "mm", while the rotating axis moves to the position specified in "deg". They will not reach their specified points simultaneously.

The speed can be specified in <expression 3> as a SPEED option, but is valid only for the DRIVE statement on this line. Do not confuse between this SPEED option and SPEED statement.

The XY option allows moving to the specified point coordinates determined by the mutual relation with the axes stated in <expression 1>. When two or more axes are specified, they will reach the specified points simultaneously. However, this option has the following limitations.

1. *The X and Y axes must be specified as a set when stated in <expression 1>.*
2. *Use this option only for Cartesian robots or SCARA robots. Do not use for other robots.*
3. *The <expression 2> and <point expression> must be stated by a real type variable or constant containing a decimal point in "mm/deg" units.*

POINT

◯ *Position data can be specified in a Cartesian coordinate system (mm/deg) or joint coordinate system (pulses). When specified in both systems, all position data is viewed as a Cartesian coordinate system to move the axes. When a Cartesian coordinate system is used to move the rotating axis, the position data is converted into degrees. However, if the XY option is specified, the axes move along the Cartesian coordinate system.*

◯ *Shift coordinates and hand coordinates are usually invalid. However, these will be valid when the XY option is selected.*

◯ *If axes other than those specified are in HOLD status, they will move when a movement command for another task is executed.*

Task n: DRIVE(1, 1000)

•
•

Task n+1: DRIVE(3, 10000)

•
•

◯ *When the XY option is not used, the specified axes will not reach the specified points simultaneously. The command will be completed when the last axis enters the OUT effective position.*

EXAMPLES:

DRIVE (1, 10000)

---------- Moves the X-axis to the "10000 pulses" position.

DRIVE (2, 10000), (3, 5000)

---------- Moves the Y-axis to the "5000 pulses" position and the Z-axis to the "5000 pulses" position.

DRIVE (3, LOCZ (P100))

---------- Moves the Z-axis to point 100 on the Z coordinate.

DRIVE (1, P0)

---------- Moves the X-axis to point 0 on the X coordinate.

DRIVE (1, P100), S=10

---------- Moves the X-axis to point 100 on the X coordinate at 10% speed.

DRIVE (1, P10), (2, P10), XY

---------- Moves the XY axes to the Cartesian coordinate position specified by point 10.

D R I V E 2 Statements

FORMAT:

DRIVE2 (<expression 1>,	<expression 2>)	[,(<expression 1>,	<expression 2>)	...]
	<point expression>		<point expression>	
		[,	SPEED	=<expression 3>][,XY]
			S	

EXPLANATION:

The DRIVE statement executes an absolute movement command for specified axes in the sub group.

Axis movement begins after positioning the arms of all axes specified in <expression 1> (to within the tolerance range), and the command terminates when the arms enter the OUT effective position. (Refer to "12-1-2 Axis Parameter" in the Controller User's Manual.)

Use the WAIT ARM2 statement (described later) for continuing the subsequent task after positioning to within the tolerance range.

Specify the axis number in <expression 1> and the position in <expression 2> or <point expression>. When <point expression> is used to specify the position, the data for the axis specified in <expression 1> is used. At this point, if the position data is in "mm/deg" units (real type variable or constant containing a decimal point), the linear axes from the "0" pulse point move to the positions specified in "mm", while the rotating axis moves to the position specified in "deg". They will not reach their specified points simultaneously.

The speed can be specified in <expression 3> as a SPEED option, but is valid only for the DRIVE2 statement on this line. Do not confuse between this SPEED option and SPEED2 statement.

The XY option allows moving to the specified point coordinates determined by the mutual relation with the axes stated in <expression 1>. When two or more axes are specified, they will reach the specified points simultaneously. However, this option has the following limitations.

4. *The X and Y axes must be specified as a set when stated in <expression 1>.*
5. *Use this option only for Cartesian robots or SCARA robots. Do not use for other robots.*
6. *The <expression 2> and <point expression> must be stated by a real type variable or constant containing a decimal point in "mm/deg" units.*

POINT

◯ *Position data can be specified in a Cartesian coordinate system (mm/deg) or joint coordinate system (pulses). When specified in both systems, all position data is viewed as a Cartesian coordinate system to move the axes. When a Cartesian coordinate system is used to move the rotating axis, the position data is converted into degrees. However, if the XY option is specified, the axes move along the Cartesian coordinate system.*

◯ *Shift coordinates and hand coordinates are usually invalid. However, these will be valid when the XY option is selected.*

◯ *If axes other than those specified are in HOLD status, they will move when a movement command for another task is executed.*

Task n: DRIVE2(1, 1000)

•
•

Task n+1: DRIVE2(3, 10000)

•
•

◯ *When the XY option is not used, the specified axes will not reach the specified points simultaneously. The command will be completed when the last axis enters the OUT effective position.*

EXAMPLES:

DRIVE2 (1, 10000)

---------- Moves the X-axis to the "10000 pulses" position.

DRIVE2 (2, 10000), (3, 5000)

---------- Moves the Y-axis to the "5000 pulses" position and the Z-axis to the "5000 pulses" position.

DRIVE2 (3, LOCZ (P100))

---------- Moves the Z-axis to point 100 on the Z coordinate.

DRIVE2 (1, P0)

---------- Moves the X-axis to point 0 on the X coordinate.

DRIVE2 (1, P100), S=10

---------- Moves the X-axis to point 100 on the X coordinate at 10% speed.

DRIVE (1, P10), (2, P10), XY

---------- Moves the XY axes to the Cartesian coordinate position specified by point 10.

D R I V E I Statements

FORMAT:

DRIVEI (<expression 1>, | <expression 2>) | <point expression> | [,(<expression 1>, | <expression 2>) | <point expression> | ...]
[, | SPEED | S | =<expression 3>]

EXPLANATION:
The DRIVEI statement is a command that moves the robot in relative units along specific axes (in the main group).
After positioning for arms of all the axes specified in <expression 1> (to within effective position tolerance), movement begins and the command terminates when the arms enter the OUT effective position (refer to "12-1-2 Axis Parameter" of the Controller User's Manual).
Use the WAIT ARM statement (described later) for continuing on to another job operation after positioning to within the tolerance area.
Use the second expression <expression 2> or the (<point expression>) to specify the position for the axis number of the first expression <expression 1>.
Axis data from <expression 1> is used when the <point expression> is used to specify the position.
Movement speed can be specified in | SPEED | S | =<expression 3>. This command is valid only to the DRIVEI statement on this line. Remember that it is different from the SPEED statement.

POINT

- *The position data can use both a cartesian coordinate system (mm/deg) and a joint coordinate system (pulses).*
  However, when moving the rotating axis in a cartesian coordinate system, the movement is converted to degrees.
- *Shift coordinates and hand definition are invalid.*
- *Since axes other than those specified are in HOLD status, they will commence movement if a movement command for another task is executed.*

  Task n: DRIVE(1, 1000)
  •
  •
  Task n+1: DRIVE(3, 1000)
  •
  •
- *If several axes were specified at the same time, then they won't arrive simultaneously. The command completes when the last axis that was out of position finally arrives.*

CAUTION

◯ ***If the robot's movement is suspended during the execution of a DRIVEI command, the position to which it will move after movement is commenced, changes.***

EXAMPLES:

D R I V E I (1 , 1 0 0 0 0)

---------- shift the X-axis to the +10,000 pulse position from current position.

D R I V E I (2 , 1 0 0 0 0) , (3 , 5 0 0 0)

---------- shift the Y-axis to the +10,000 pulse position from current position and the Z-axis to the +5,000 pulse position from current position.

D R I V E I (3 , P 1 0 0)

---------- shift to the Z-axis (current position + Z-axis coordinate P100).

D R I V E I (2 , P 1 0) , S = 5 0

---------- shift to the Y-axis (current position + Y-axis coordinate P10) at the speed of 50%.

RELATED COMMAND: WAIT ARM

D R I V E I 2 Statements

FORMAT:

DRIVEI2 (<expression 1>, | *<expression 2>)* / *<point expression>* | *[,(<expression 1>,* | *<expression 2>)* / *<point expression>* | *...]*

[, | *SPEED* / *S* | *=<expression 3>]*

EXPLANATION:

The DRIVEI2 statement is a command that moves the robot in relative units along specific axes (in the sub group).

After positioning for arms of all the axes specified in <expression 1> (to within effective position tolerance), movement begins and the command terminates when the arms enter the OUT effective position (refer to "12-1-2 Axis Parameter" of the Controller User's Manual).

Use the WAIT ARM2 statement (described later) for continuing on to another job operation after positioning to within the tolerance area.

Use the second expression <expression 2> or the (<point expression>) to specify the position for the axis number of the first expression <expression 1>.

Axis data from <expression 1> is used when the <point expression> is used to specify the position.

Movement speed can be specified in | *SPEED* / *S* | *=<expression 3>. This command is valid only to the DRIVEI2 statement on this line. Remember that it is different from the SPEED2 statement.*

CAUTION

- ***This command is valid only when the sub group has been set in system generation.***
- ***If the robot's movement is suspended during the execution of a DRIVEI2 command, the position to which it will move after movement is commenced, changes.***
- ***When a cartesian coordinate system (mm/deg) is used as the position data, shift coordinates and hand coordinates are invalid. However, when moving the rotating axis in a cartesian coordinate system, the movement is converted to degrees.***

EXAMPLES:

D R I V E I 2 (1 , 1 0 0 0 0)

---------- shift the X-axis to the +10,000 pulse position from current position.

D R I V E I 2 (2 , 1 0 0 0 0) , (3 , 5 0 0 0)

---------- shift the Y-axis to the +10,000 pulse position from current position and the Z-axis to the +5,000 pulse position from current position.

D R I V E I 2 (3 , P 1 0 0)

---------- shift to the Z-axis (current position + Z-axis coordinate P100).

D R I V E I 2 (2 , P 1 0) , S = 5 0

---------- shift to the Y-axis (current position + Y-axis coordinate P10) at the speed of 50%.

RELATED COMMAND: WAIT ARM2

EXIT FOR Statements

FORMAT:

```
EXIT FOR
```

EXPLANATION:
This completes the FOR and NEXT loop and then jumps to the following command of the nearest NEXT statement.

EXAMPLE:

```
'INPUT TESTER WITH STOP SIGNAL
FOR A=&B00000001 TO &B00001111
    WAIT DI3()=A
    PRINT "A=";A
    IF DI(40)=1 THEN EXIT FOR
NEXT
HALT
```

RELATED COMMAND: FOR, NEXT

EXIT SUB Statements

FORMAT:

```
EXIT SUB
```

EXPLANATION:
This command exits the subroutine and returns to the line after the CALL statement.

POINT

This command is effective only for subroutines defined by SUB and END SUB statements.

EXAMPLE:

```
CALL *SUB1(REF X%, REF Y%)
HALT
SUB *SUB1(X%, Y%)
IF X%>=Y% THEN EXIT SUB
TEMP%=Y%
X%=Y%
Y%=TEMP%
END SUB
```

RELATED COMMAND: CALL, SUB, END SUB

EXIT TASK Statements

FORMAT:

```
EXIT TASK
```

EXPLANATION:
This command ends the task that it is included in.

POINT

- *This command may not be used in task 1.*
- *To use this command, a high-speed arithmetic processor (option) must be provided.*

EXAMPLE:

```
        START *SUBTASK, T2
    *L0:
        MOVE P, P0
        MOVE P, P1
        GOTO *L0
    *SUBTASK:
        WAIT DI2(0)=1
        DO2(0)=1
        DELAY 1000
        DO2(0)=0
        WAIT DI2(0)=0
        EXIT TASK---------- SUBTASK is ended.
```

RELATED COMMAND: CUT, RESTART, START, SUSPEND

FOR and NEXT Statements

These command statements are used to create loops to repeat procedures in a program.

FORMAT:

```
FOR <control variable>=<expression 1> TO <expression 2> [STEP
    <expression 3>]
        :
NEXT [<control variable>]
```

EXPLANATION:
The routine included between the FOR and NEXT statements is repeated the number of times defined by the difference between <expression 1> and <expression 2>, calculated in the steps defined by <expression 3>. The value of <expression 3> may be negative.
The value of the <control variable> may be expressed by a simple variable or an array variable.

POINT

If the value of <expression 3> is 1, STEP<expression 3> may be eliminated.

EXAMPLES:

```
'CYCLE WITH CYCLE NUMBER OUTPUT TO DISPLAY
FOR A=1 TO 10
    MOVE P, P0
    MOVE P, P1
    MOVE P, P2
    PRINT "CYCLE NUMBER=";A
NEXT A
HALT
'INPUT TESTER
FOR A=&B00000001 TO &B00001111
    WAIT DI3()=A
    PRINT "A=";A
NEXT A
HALT
```

RELATED COMMAND: EXIT FOR

GOSUB and RETURN Statements

These statements are used to call subroutines and return to the original program after the subroutine has been executed.

FORMAT:

```
        | GOSUB  |  <label>
        | GO SUB |
        •••••••
        •••••••
<label> :
              RETURN
```

EXPLANATION:
The robot will jump to the subroutine defined by the <label> in the GOSUB statement, and execute the commands therein. When a RETURN statement is found, the robot will return the line after the GOSUB statement.

POINT

40 GOSUB statements may be used successively.

EXAMPLE:

```
'CYCLE WITH MESSAGES
*L1:
    A=0
    MOVE P, P[A]
    GOSUB *MESSAGES
    A=1
    MOVE P, P[A]
    GOSUB *MESSAGES
GOTO *L1
*MESSAGES:
    PRINT "POSITION:";P[A]
    RETURN
```

RELATED COMMAND: DECLARE

CAUTION

To end a subprocedure to which a jump was made with the GOSUB statement, always use the RETURN statement.

If another statement such as GOTO is used to exit from the subprocedure, an error such as "Stack Overflow" may occur.

G O T O Statements

FORMAT:

G O T O *G O T O*	*<label>*

EXPLANATION:
This statement will cause the robot to jump unconditionally to the <label> specified.

EXAMPLE:

```
*L1:
    MOVE P, P1
    MOVE P, P2
    GOTO *L1
```

H A L T Statements

FORMAT:

H A L T	*[*	*<expression>* *<character string>*	*]*

EXPLANATION:

This statement will stop the execution of the program and reset the robot. When execution begins again, the program is run from its beginning.

If an <expression> or <character string> is included in the statement, it is displayed in the appropriate format when the statement takes effect.

EXAMPLES:

```
H A L T
H A L T  C O U N T
H A L T  " P R O G R A M  S T O P "
```

H A N D Definition Statements, C H A N G E Statements (Hand Selection for Main Robot)

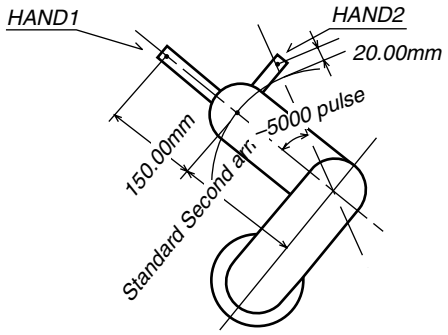
This command is used to define or change the hand position for the main robot.
FORMAT 1: For SCARA Robots

Definition Statement:
H A N D H n =<1st parameter><2nd parameter><3rd parameter>[R]
Selection Statement:
C H A N G E H n

n: 0 to 3

a. *If the 4th parameter "R" is not specified, the hand installed on the second arm's end is selected (see figure).*
   <1st parameter>: The offset value in pulses between the standard second arm position and the imaginary second arm of hand "n". The "+" direction is counterclockwise.
   <2nd parameter>: The difference between the length of the imaginary second arm of hand "n" and the standard second arm in millimeters.
   <3rd parameter>: The Z-axis offset value for hand "n" in millimeters.

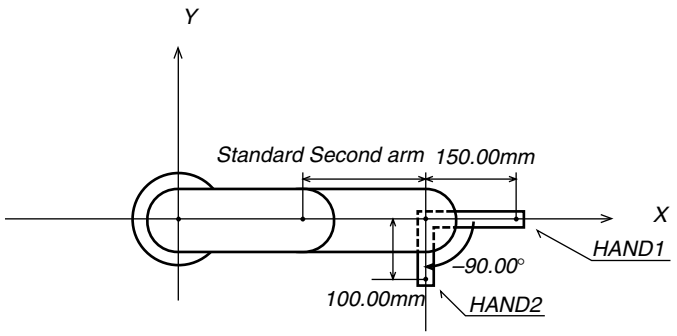
   EXAMPLE:
   H A N D H 1 = 0 1 5 0 . 0 0 0 . 0 0
   H A N D H 2 = - 5 0 0 0 2 0 . 0 0 0 . 0 0



b. *If the 4th parameter "R" is specified, it defines the offset amount of the hand from the center of revolution of the R-axis if the R-axis is a servo motor (see figure).*
   <1st parameter>: When the position of the R-axis is 0.00, this is the angle of hand "n" from the X-axis of a Cartesian coordinate system. The "+" direction is the counterclockwise direction.
   <2nd parameter>: The length of hand "n" in millimeters (>0).
   <3rd parameter>: The Z-axis offset amount for hand "n" in millimeters.

EXAMPLE:

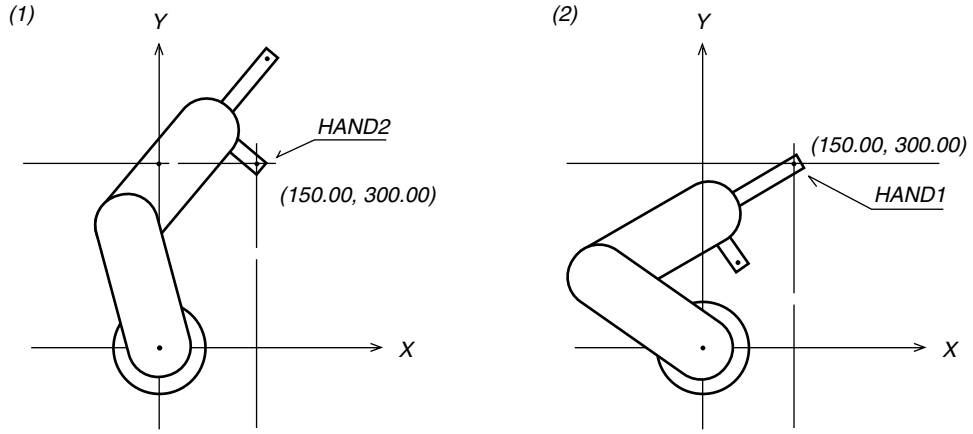
```
HAND H1=      0.00  150.00     0.00          R
HAND H2=    -90.00  100.00     0.00          R
```



EXAMPLES:

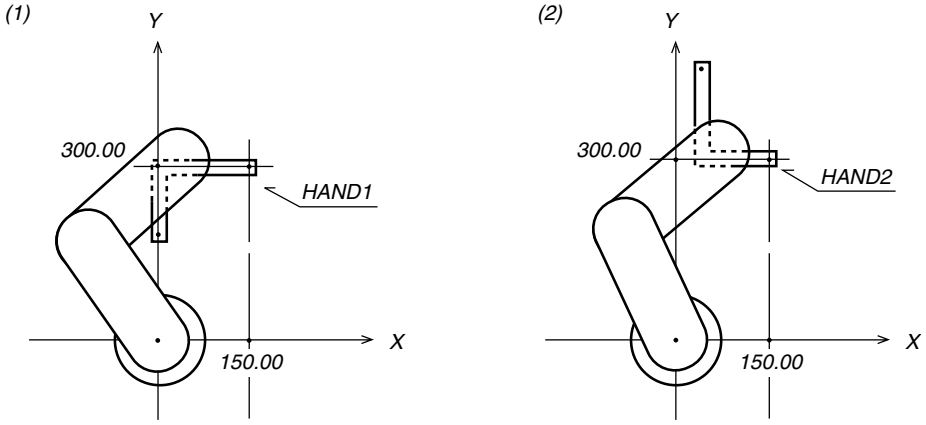
a. When <4th parameter> R is not specified

```
HAND H1=       0   150.0     0.0
HAND H2=   -5000    20.0     0.0
P1=150.00 300.00    0.00    0.00    0.00    0.00
CHANGE H2
MOVE P, P1 ---------- Hand 2 moves to point P1. (1)
CHANGE H1
MOVE P, P1 ---------- Hand 1 moves to point P1. (2)
```



b. When <4th parameter> R is specified

```
HAND H1=     0.00  150.0     0.0        R
HAND H2= -90.00100.00     0.0        R
P1=150.00 300.00   0.00   0.00   0.00   0.00
CHANGE H1
MOVE P, P1 ---------- Hand 1 moves to point P1. (1)
CHANGE H2
MOVE P, P1 ---------- Hand 2 moves to point P1. (2)
```



FORMAT 2: For Cartesian coordinate system robots

Definition Statement:

HAND Hn =<1st parameter><2nd parameter ><3rd parameter >[R]

Selection Statement:

CHANGE Hn

n: 0 to 3

a. When the <4th parameter > R is not specified, the hand installed on the tip of the second arm is assumed (see figure).

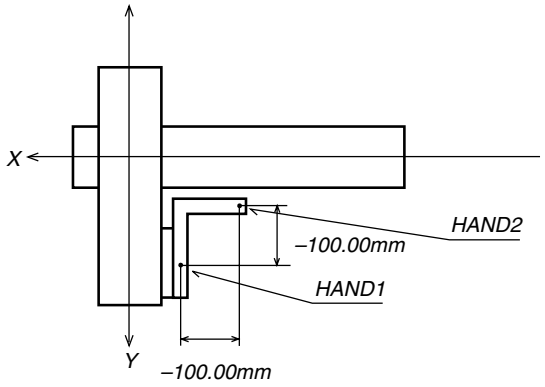
<1st parameter >: Hand "n" X-axis offset amount in millimeters.

<2nd parameter >: Hand "n" Y-axis offset amount in millimeters.

<3rd parameter >: Hand "n" Z-axis offset amount in millimeters.

EXAMPLE:

```
HAND H1=      0.00     0.00     0.00
HAND H2=  -100.00 -100.00 -100.00
```



b. When the <4th parameter > R is specified, it represents the hand offset from the R-axis center of revolution if it is a servo motor.

<1st parameter >: When the R-axis position in 0.00 this is the angle between the hand "n" and the X-axis of a Cartesian coordinate system. The "+" direction is counterclockwise.

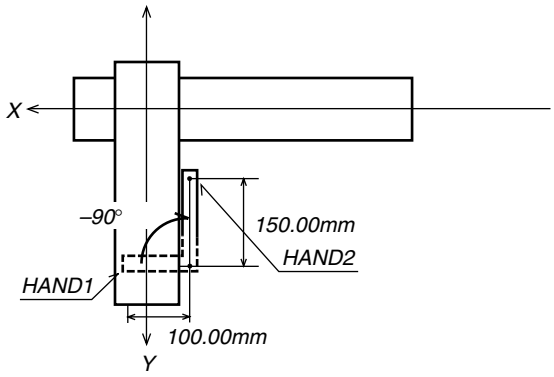
<2nd parameter >: The length of hand "n" in millimeters (>0).

<3rd parameter >: The Z-axis offset amount for hand "n" in millimeters.

EXAMPLE:

```
HAND H1=      0.0   100.00      0.00    R
HAND H2=    -90.0   150.00   -100.00    R
```

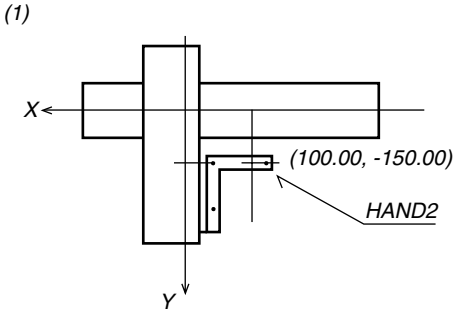
(R-axis position is 0.00)

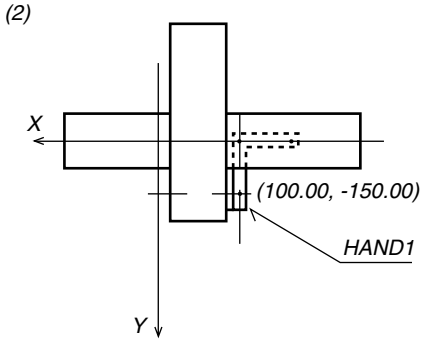


EXAMPLES:

a. When the <4th parameter > R is not specified

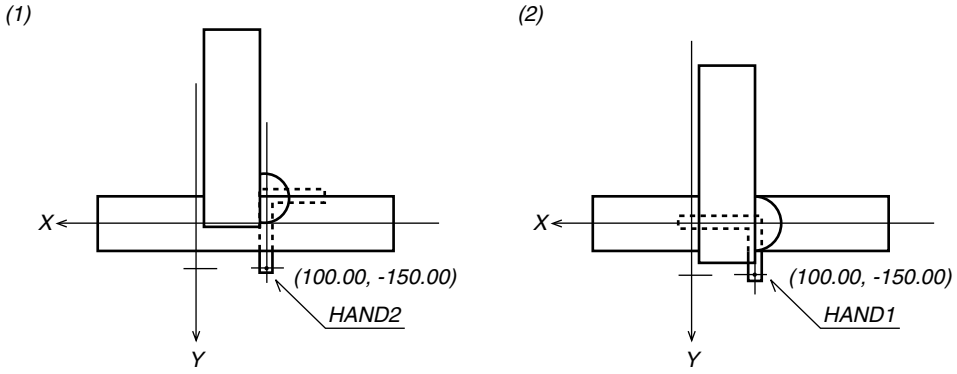
```
HAND H1=      0.0      0.0      0.0
HAND H2=  -100.00  -100.00  -100.00
P1=-150.00 100.00   0.00   0.00   0.00   0.00
CHANGE H2
MOVE P, P1 ---------- Hand 2 moves to point P1. (1)
CHANGE H1
MOVE P, P1 ---------- Hand 1 moves to point P1. (2)
```





b. When <4th parameter > R is specified

```
HAND H1=     0.00  100.00    0.00        R
HAND H2=   -90.00  150.00    0.00        R
P1=-150.00 100.00    0.00   90.00    0.00    0.00
CHANGE H2
MOVE P, P1 ---------- Hand 2 moves to point P1. (1)
CHANGE H1
MOVE P, P1 ---------- Hand 1 moves to point P1. (2)
```



CAUTION

When the power is turned off during execution of a hand definition statement, it may cause a "9.7: Hand data destroyed" message to be issued.

H A N D 2 Definition Statements, C H A N G E 2 State- ments (Hand Selection for Sub Robot)

This command is used to define or change the hand position for the sub robot. This command is valid only when the sub group has been set in system generation.

FORMAT 1: For SCARA Robots

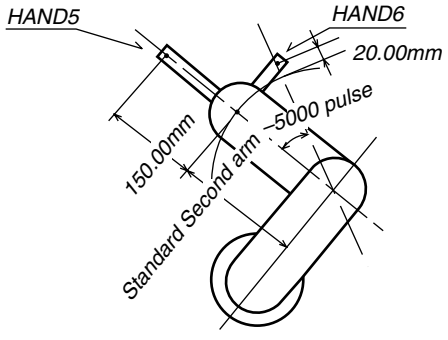
Definition Statement:
H A N D 2 H n =<1st parameter><2nd parameter><3rd parameter>[R]
Selection Statement:
C H A N G E 2 H n

n: 4 to 7

a. *If the 4th parameter "R" is not specified, the hand installed on the second arm's end is selected (see figure).*
   <1st parameter>: The offset value in pulses between the standard second arm position and the imaginary second arm of hand "n". The "+" direction is counterclockwise.
   <2nd parameter>: The difference between the length of the imaginary second arm of hand "n" and the standard second arm in millimeters.
   <3rd parameter>: The Z-axis offset value for hand "n" in millimeters.

EXAMPLE:

```
H A N D 2  H 5 =        0   1 5 0 . 0 0      0 . 0 0
H A N D 2  H 6 = - 5 0 0 0    2 0 . 0 0      0 . 0 0
```



b. *If the 4th parameter "R" is specified, it defines the offset amount of the hand from the center of revolution of the R-axis if the R-axis is a servo motor (see figure).*

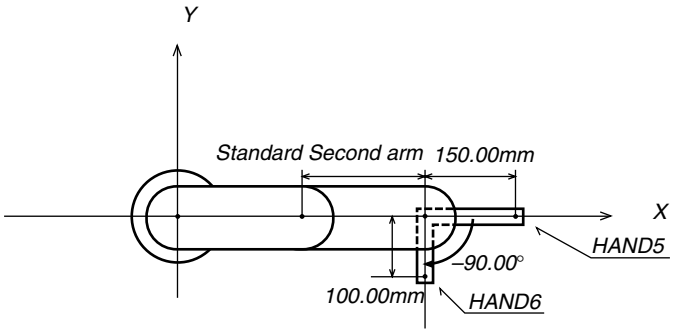
<1st parameter>: When the position of the R-axis is 0.00, this is the angle of hand "n" from the X-axis of a Cartesian coordinate system. The "+" direction is the counterclockwise direction.

<2nd parameter>: The length of hand "n" in millimeters (>0).

<3rd parameter>: The Z-axis offset amount for hand "n" in millimeters.

EXAMPLE:

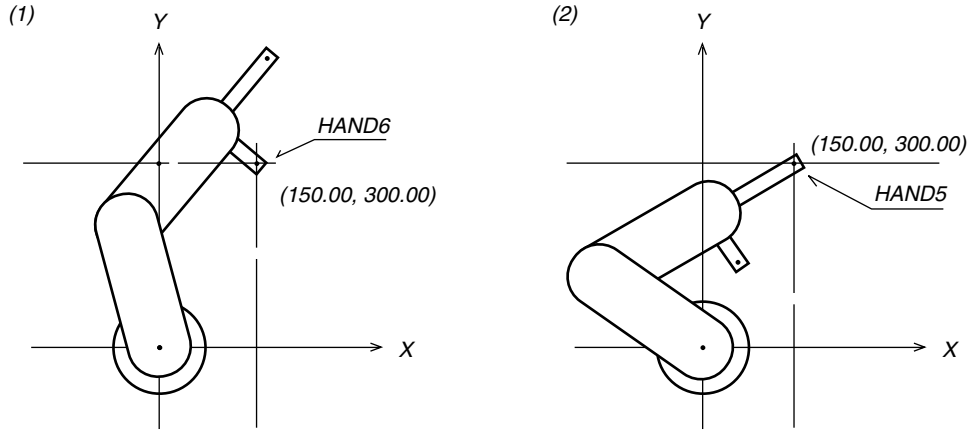
```
HAND2 H5=      0.00  150.00     0.00        R
HAND2 H6=    -90.00  100.00     0.00        R
```



EXAMPLES:

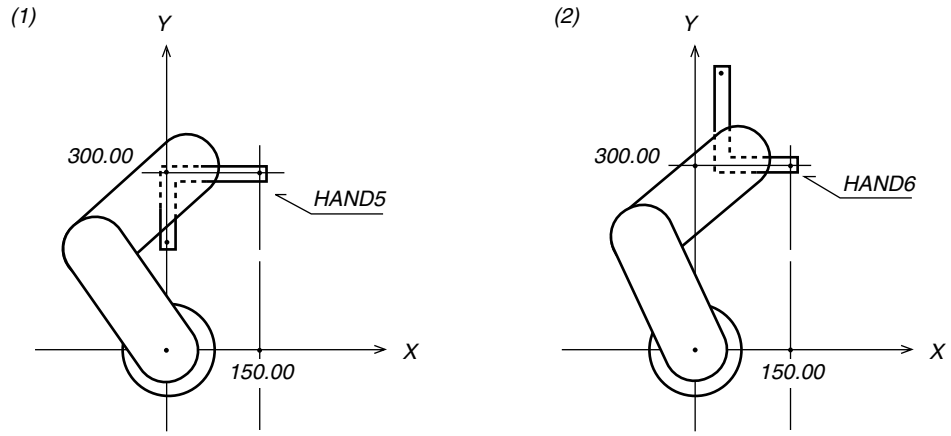
a. When <4th parameter> R is not specified

```
HAND2 H5=    0     150.0    0.0
HAND2 H6=    -5000    20.0      0.0
P1=    150.00    300.0    00.00    0.00    0.00    0.00
CHANGE2 H6
MOVE2 P, P1 ---------- Hand 6 moves to point P1. (1)
CHANGE2 H5
MOVE2 P, P1 ---------- Hand 5 moves to point P1. (2)
```



b. When <4th parameter> R is specified

```
HAND2 H5=    0.00   150.0      0.0        R
HAND2 H6=-90.00   100.00      0.0        R
P1=150.00  300.00    0.00    0.00    0.00    0.00
CHANGE2 H5
MOVE2 P, P1 ---------- Hand 5 moves to point P1. (1)
CHANGE2 H6
MOVE2 P, P1 ---------- Hand 6 moves to point P1. (2)
```



FORMAT 2: For Cartesian coordinate system robots

Definition Statement:

```
HAND2 Hn =<1st parameter><2nd parameter ><3rd parameter >[R]
```

Selection Statement:

```
CHANGE2 Hn
```

n: 4 to 7

a. *When the <4th parameter > R is not specified, the hand installed on the tip of the second arm is assumed (see figure).*

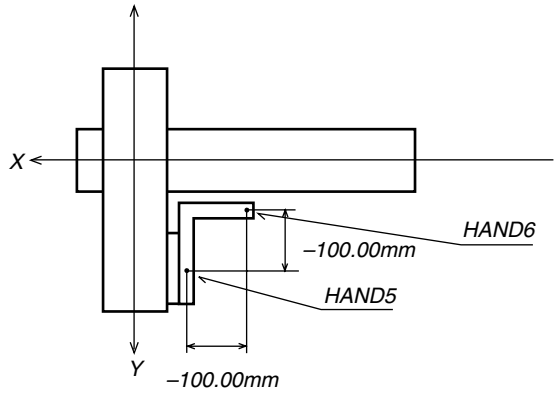
<1st parameter >: Hand "n" X-axis offset amount in millimeters.

<2nd parameter >: Hand "n" Y-axis offset amount in millimeters.

<3rd parameter >: Hand "n" Z-axis offset amount in millimeters.

EXAMPLE:

```
HAND2 H5=      0.00      0.00      0.00
HAND2 H6= -100.00 -100.00 -100.00
```



b. *When the <4th parameter > R is specified, it represents the hand offset from the R-axis center of revolution if it is a servo motor.*

<1st parameter >: When the R-axis position in 0.00 this is the angle between the hand "n" and the X-axis of a Cartesian coordinate system. The "+" direction is counterclockwise.

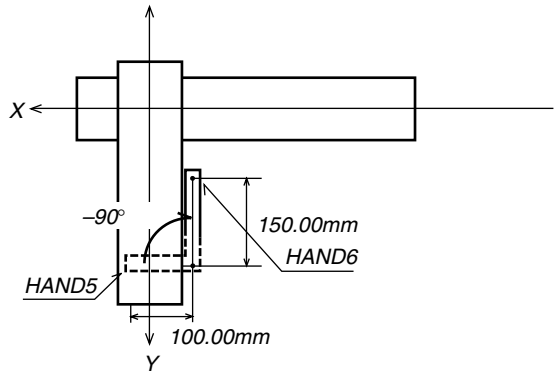
<2nd parameter >: the length of hand "n" in millimeters (>0).

<3rd parameter >: The Z-axis offset amount for hand "n" in millimeters.

EXAMPLE:

```
HAND2 H5=   0.0   100.00      0.00      R
HAND2 H6=-90.0   150.00 -100.00      R
```

(R-axis position is 0.00)



H O L D Statements

FORMAT:

H O L D	*[*	*<expression>* *<character string>*	*]*

EXPLANATION:

This command stops the program temporarily. When execution is resumed, it begins from the line of the program following the HOLD statement.

POINT

If an <expression> or <character string> is specified, it is displayed in the appropriate format when the HOLD statement is executed.

EXAMPLES:

```
H O L D
H O L D  C O U N T
H O L D  " E R R O R  S T O P "
```

I F Statements

FORMAT 1:

```
I F <expression> T H E N |<label 1>                |[ELSE|<label 2>                |]
                         |<command statement 1>|     |<command statement  2>|
```

EXPLANATION:
When the conditions of the <expression> in the command are met, the robot will jump to the line specified by the <label 1>. Or, <command statement 1> will be executed. If the condition is not met, the robot proceeds to the next line. If an ELSE condition has been specified, <label 2> is jumped to when the condition is not met. Or, the <command statement 2> is executed.

EXAMPLE:

```
'L I M I T E D  P U L S E  C O U N T E R
    L I M I T = 1 0 0 0
*L 1 :
    I F  D I 3 ( ) = & B 0 0 0 0 0 0 0 1  T H E N  A = A + 1
    I F  A  <  L I M I T  T H E N  *L 1  E L S E  H A L T
```

FORMAT 2:

```
I F  <expression>  T H E N

    Block 1

[ E L S E ]

    Block 2

E N D I F
```

EXPLANATION:
If the conditions of the <expression> are met, this command will execute block 1.
If there is and ELSE condition specified, and the condition is not met, the robot will move to block 2.

EXAMPLE:

```
IF DI3(1)=1 THEN
    MOVE P, P1
    DO(30)=1
    DELAY 100
ELSE
    MOVE P, P2
    DO(31)=1
    DELAY 100
ENDIF
HALT
```

I N P U T Statements

FORMAT:

I N P U T [<prompt>]	*; ,]*	*<variable>* *<point variable>* *<shift variable>*	*[,*	*<variable>* *<point variable>* *<shift variable>*	*· · ·]*

EXPLANATION:

- *The INPUT statement will change the operation mode of the robot temporarily to MPB mode to open it to input from the user.*
- *The <prompt> is the message that will be displayed when this command is executed and the robot is waiting for input data.*
- *If the <prompt> is followed by a semicolon (;), a question mark (?) and a space will be displayed after the prompt. If a comma (,) is placed after the <prompt>, nothing will be displayed after the prompt.*
- *If the <prompt> is eliminated, the robot will display a question mark (?) and a space by itself while waiting for input.*
- *The data input from the MPB will be assigned to the <variable>.*
- *More than one variable may be specified for input. Each variable must be separated by a comma (,) in the command statement itself and when the data is input from the console.*
- *The format of the data that is actually input must match the format of the variables specified in the command. Especially in the case of point and shift variables, the number of axes in the data must be correct.*

EXAMPLE:

```
I N P U T  P 0
? 100.00  100.00  100.00  100.00  100.0  100.0 0 c/r
```

The underlined part is the input data.

- *If no input is given after a prompt, and only a carriage return is input, the robot will treat the entry as a null value (0) or null string (a string with no characters). In this way it is possible to input null values or strings, but if the INPUT statement expects more than one variable to be input, then a comma must be input for every variable except the last.*
- *If a character string is to be input, it is not necessary to surround the string with double quotation marks ("). However, if you wish to include characters in the string that have a special meaning, such as a space at the head of the string, it is necessary to begin and end the string with quotation marks.*
- *If you do not wish to enter any data, press the [STOP] key. The execution of the program will stop. When the program is run again, the INPUT statement will once again be executed.*
- *If you do not wish to stop the program, but also want to skip the input for the INPUT command, press the [ESC] (escape) key.*

POINT

◯ *If the format of the input does not match the format of the variables specified in the command, an error message "? Read from start" will be displayed after the question mark, asking for repeated input. The robot will wait for the correct input.*

EXAMPLES:

```
INPUT A
INPUT "FILE NAME=";F_NAME$
INPUT "P10=";P10
INPUT S0, S1
```

RELATED COMMAND: PRINT, SEND

L E T Statements (Assigning Values to Variables)

FORMAT:

[LET]	*<arithmetical expression>* *<character string assignation expression>* *<point value assignation expression>* *<shift value assignation expression>*	

● *Arithmetical Expressions*

FORMAT:

[LET]	*<arithmetical variable>* *<output variable>* *<point element variable>* *<shift element variable>* *<internal output variable>*	*=<expression>*

EXPLANATION:
The variable on the left of the equal sign in the <expression> is assigned the value on the right.

EXAMPLES:

```
A = B + 1
B ( 1 ,  2 ,  1 ) = 1 0 . 0 5
D O 2 ( ) = A
L O C X ( P 1 0 ) = A ( 1 )
L O C Z ( S 0 ) = 1 0 0 . 0 0
M O ( 3 7 ,  2 5 ,  2 0 ) = & B 1 0 1
```

● *Character String Assignment Expressions*

FORMAT:

[L E T]<character string variable>=<character string expression>

EXPLANATION:
The result of the <character string expression> is assigned to the character variable.

POINT

The only arithmetical symbol that can be used in character string expressions is the "+" sign. No parentheses may be used.

EXAMPLE:

```
A $ = " Y A M A H A "
B $ = " M O U N T E R "
C $ = " R O B O T "
D $ = A $ + "  " + B $
E $ = A $ + "  " + C $
P R I N T  D $
P R I N T  E $
```

Results in:

```
Y A M A H A  M O U N T E R
Y A M A H A  R O B O T
```

- ***Point Value Assignation Expressions***

FORMAT:

[L E T] <point variable>=<point expression>

EXPLANATION:
The value of the <point expression> is assigned to the <point variable>. Point expressions may use the arithmetical symbols: "+", "-", "", "/". Functions can also be used. (Refer to "10 Functions".)*

EXAMPLES:

```
P 1 = P 1 0 0
```
---------- Point 100's coordinates are assigned to point 1.

```
P 2 0 0 = P 2 0 0 + P 5
```
---------- The coordinates of point 200 and point 5 are added to each other respectively, and the result is assigned to point 200.

```
P 3 0 0 = P 3 0 0 - P 3
```
---------- The coordinates of point 3 are subtracted from those of point 300 respectively, and the result is assigned to point 300.

```
P 8 0 = P 7 0 * A
```
---------- Each coordinate of point 70 is multiplied by the value of A and the result is assigned to point 80.

```
P 6 0 = P 5 / 3
```
---------- Each coordinate of point 5 is divided by 3 and the result is assigned to point 60.

POINT

◯ *The arithmetical symbols used in point expressions are: " * ", " / ", " + ", and " - ". " * " and " / " are limited to multiplication and division by constant or variable. They cannot be used to multiply and divide by other point variables.*

EXAMPLES:

```
P 1 5 * 5 --------- Possible
P [ E ] / 3 --------- Possible
P 1 0 * P 1 2 ---- Impossible
3 / P 1 0 ---------- Impossible
```

◯ *The result of point variable multiplication and division by constant or variable depends upon the type of point data.*

◯ *The result of joint coordinate expressions is converted into integers before being assigned to a point variable.*

EXAMPLE:

```
P 0 = 1 0 0   2 0 0   3 0 0   4 0 0   5 0 0   6 0 0
P 1 = P 0 / 2 . 3 3
    ---------- P 1 = 4 2   8 5   1 2 8   1 7 1   2 1 4   2 5 7
```

◯ *The result of Cartesian coordinate expressions is rounded off at the third dicimal place and assigned to the point variable.*

EXAMPLE:

```
P 0 = 1 0 0 . 0 0  2 0 0 . 0 0  3 0 0 . 0 0  4 0 0 . 0 0  5 0 0 . 0 0  6 0 0 . 0 0
P 1 = P 0 / 2 . 3 3
    ---------- P1= 42.92  85.84  128.76  171.67  214.59  257.51
```

CAUTION

When the power is turned off during execution of a point value assignation statement, it may cause a "9.2: Point data destroyed" message to be issued.

● *Shift Assignment Expressions*

FORMAT:

```
[ L E T ]<shift variable>=<shift expression>
```

EXPLANATION:
The value of the <shift expression> is assigned to the <shift variable>. The only arithmetical symbols that may be used in shift variable expressions are the " + " sign and the " - " sign. Parentheses may not be used.

EXAMPLE:

S 2 = - S 0 + S 1

---------- The sign of each coordinate of the shift data of shift number 0 is reversed and added to its corresponding element of shift number 1. The result is assigned to shift number 2.

POINT

Only shift numbers may be used in shift expressions.
S0=S0+A(1) Impossible

CAUTION

When the power is turned off during execution of a shift value assignation statement, it may cause a "9.6: Shift data destroyed" message to be issued.

L O Statements (Arm lock)

FORMAT:

[L E T]	*L O m ([b , • • •, b])* *L O (m b , • • •, m b)*	*=<expression>*

m: Port numbers 0
b: Bit definition 0 to 7

EXPLANATION:
This command statement prohibits an axis movement or cancels it. LO(00) to LO(07) corresponds to axis 1 through axis 8. An axis movement is prohibited by ON status.

POINT

Be sure to define bits in ascending order from the right.

EXAMPLES:

L O ( ) = & B 0 0 0 0 1 0 1 0
---------- Movement of the 2nd axis and 4th axis are prohibited.

L O 0 (5 , 2 , 0) = & B 1 0 1
---------- Movement of the 6th axis and 1st axis are prohibited.

L O (0 7 , 0 4 , 0 2 , 0 1) = A
---------- The 4 lower bits value of the variable A are assigned to LO(07), (04), (02) and (01).

RELATED COMMAND: RESET, SET

M O Statements (Internal Output)

FORMAT:

[L E T]	*M O m ([b , • • •, b])* *M O (m b , • • •, m b)*	*=<expression>*

m: Port numbers 2 to 7, 10 to 13
b: Bit definition 0 to 7

EXPLANATION:
The specified value is output to the MO. If the arm is moving, this output is available when the movement is complete (the axis has reached the target position.)

POINT

- *It is not possible to output to MO0 and MO1.*
- *Bit 0 of MO0 maintains status of axis origin sensors 1 to 8 (in order).*
- *The bit is "1" when the origin sensor is ON and is "0" when the origin sensor is OFF.*
- *MO1 is for hold status of axes 1 to 8 (in order from bit 0). "1" is hold, "0" is nonhold. When the servo is set to OFF, the status is nonhold. The non-use axis sets to "1".*
  HOLD is the status for times when shifted with the MOVE command and placed within the tolerance for the target position.

EXAMPLES:

M O 2 ( ) = & B 1 0 1 1 1 0 0 0
---------- MO(23), (24), (25), (27) are ON, MO(20), (21), (22), (26) are OFF.

M O 2 (6 , 5 , 1) = & B 0 1 0
---------- MO(25) is ON, MO(21), (26) are OFF.

M O 2 ( ) = 1 5
---------- MO(20), (21), (22), (23) are ON, MO(24), (25), (26), (27) are OFF.

M O (3 7 , 3 5 , 2 7 , 2 0) = A
---------- The 4 lower bits of the variable A are output to MO(37), (35), (27), (20).

RELATED COMMAND: RESET, SET

M O V E Statements

FORMAT:

M O V E	*P T P* *P* *L* *C*	*, <point definition>[, option [, option, , ,]]*

EXPLANATION:
This statement executes moving commands for the main robot. Therefore, only axes which are set as the main robot axes can be moved with this command. Sub robot axis and auxiliary axes cannot be moved.

Movement type : PTP, linear interpolation, circular interpolation
Point definition : Cartesian coordinates, point definition
Option : Speed setting, arch motion setting, STOPON conditions setting

Movement types

- *PTP (Point to Point)*
  *After confirming that a moving axis is placed in the tolerance*1 of the current position, movement begins and then the command terminates when the arms enters the OUT effective position*2 and then execution of the next command starts.*
  Use WAIT ARM statements (described later) when waiting for positioning to within the tolerance area.

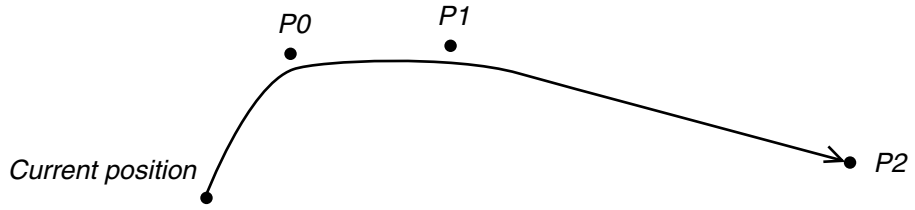
  EXAMPLE:
  M O V E P T P , P 0 and M O V E P , P 0
  ---------- Moves from current position to P0.

- *Linear interpolation (Three-dimensional)/Circular interpolation (Two-dimensions on XY plane)*
  After confirming that the moving axis is placed in the tolerance of the current position, movement begins, and execution of the next command starts immediately afterwards. Therefore, when DO is output right after linear interpolation, this is executed right after start of movement. Movement is smooth if linear interpolation or circular interpolation commands are continuously executed.
  Use WAIT ARM statements (described later) for waiting for positioning to within the tolerance area.

EXAMPLE:

MOVE L, P0, P1, P2, S=10

---------- Passes near P0 and P1 from current position at 10% speed and moves to P2.

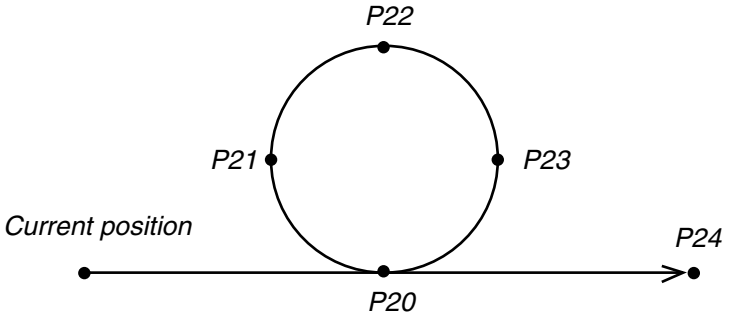


In case of circular interpolation, one circular arc is formed from 3 positions; the current position + intermediate position + target position. Therefore the number of point definitions is specified by an even number.
When a circle is drawn, at least 4 points are necessary in the following figure.

EXAMPLE:

MOVE L, P20
MOVE C, P21, P22, P23, P20
MOVE L, P24

---------- Moves in a straight line from the current position to P20, then moves in an arc formed by points P20, P21, P22 and P23, and moves in a straight line to P24.



Point definition types

- *Cartesian coordinates*

FORMAT:

```
X  Y  Z  R  A  B
```

X, Y, Z, R, A, B are the coordinates values of each axis which is separated by spaces.

EXPLANATION:
Specifies the Cartesian coordinates.

EXAMPLE:

```
MOVE P, 1000 1000 1000 1000 0 0
        ---------- Moves to the specified position by PTP control.
```

POINT

◯ *For Ver.5.09 and earlier, cannot use cartesian coordinates.*
◯ *Cartesian coordinates cannot be used with circular interpolation.*

- *Point definition*

FORMAT:

```
<point expression>[, point expression>, •••, <point expression>]
```

EXPLANATION:
◯ *Sets a point number.*
◯ *Point definition may contain many points, separated by "," commas.*

EXAMPLE:

```
MOVE P, P110+P111
        ---------- Moves from the current position to the position
                   which is calculated by P110+P111.
MOVE P, P[A]
        ---------- Moves from current position to P[A].
```

POINT

Circular interpolation must always specified with an even number.

Option types

- *Speed setting 1*

FORMAT:

SPEED *S*	*= <expression>*

The value of <expression> is 1 to 100.

EXPLANATION:

◯ *The ratio for the robot (characteristic) maximum speed is set by <expression>.*

◯ *This setting is valid only for the specified MOVE statement.*

EXAMPLE:

```
MOVE P, P0, P1, P2, S=10
     ---------- Moves at 10% speed in the sequence; P0, P1, P2.
```

- *Speed setting 2*

FORMAT:

VEL	*=<expression>*

The value of <expression> is 1 to 500 in the case of SCARA type robots.
The value of <expression> is 1 to 1000 in the case of XY type robots.

EXPLANATION:

◯ *Set XY max. composite speed of the robot by <expression> in mm/sec. units.*

◯ *Valid only for the specified MOVE statement and interpolation movement.*

◯ *Multi-type robots or loader type robots cannot be used.*

◯ *Except for XY-axis, moves at (<expression>/10)% for a max. axis speed.*

EXAMPLE:

```
MOVE L, P0, P1, P2, VEL=10
     ---------- XY composite speed is 10mm/sec and moves in
                the sequence of; P0, P1, P2.
```

POINT

For Ver.5.22 and earlier, speed setting 2 cannot be used.

- *Arch motion setting*

FORMAT:

x=<expression>[, x=<expression>,•••]

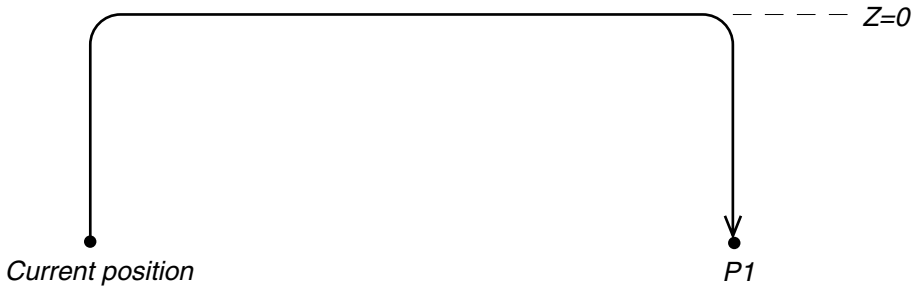
In the expression, x specifies the X, Y, Z, R, A, B axes.
When the <expression> is in real numbers, the units are in mm. When it is in integers, the units are in pulses.

EXPLANATION:

◯ *First, the X-axis moves to <expression> position, then other axes move to the target position, and finally the X-axis to the target position.*
◯ *The arch motion can only be specified for PTP movement.*
◯ *Execution of arch motion including the X or Y-axis is possible, when the target position or the value of <expression> is in integers in the SCARA type robot, and when the value of <expression> is integers in the XY type robot.*
◯ *The arch starting position can be changed by arch motion position*3.*

EXAMPLE:

M O V E P , P 1 , Z = 0
---------- First, the Z-axis moves to 0 pulse, other axes moves to P1, and finally Z-axis moves to P1.



- *STOPON conditions setting*

FORMAT:

```
STOPON  <condition expression>
```

EXPLANATION:

◯ *Specify where you wish to stop when the <condition expression> are setup. When conditions are set during the start of movement, the next command is executed without movement.*

◯ *When PTP or linear interpolation is set, the STOPON conditions setting can be used. In such a case, the next command is executed after entering within tolerance of the target position in linear interpolation.*

◯ *Only program execution is valid.*

EXAMPLE:

```
MOVE P, P100, S=70, STOPON DI(31)=1
```

---------- *Moves at 70% speed to P100. During movement to DI(31)=1, and then stops.*

EXAMPLE:

<Example 1>

```
FOR A=1 TO 10
    MOVE P, P[A]
NEXT A
```

---------- *Moves from current position in the sequence of; P1, P2, P3, •••, P10.*

<Example 2>

```
FOR A=1 TO 10
    DO(30)=0
    MOVE P, P0, Z=10000
    DO(30)=1
    DELAY 500
    MOVE P, P[A], Z=0
    DO(30)=0
    DELAY 500
NEXT A
```

---------- *Moves from work feed position P0 to P1, P2, P3, •••, P10.*

- *Acceleration setting*

FORMAT:

```
A C C = <expression>
```

The value of <expression> is 1 to 100.

EXPLANATION:

◯ *Specifies the ratio to the current robot acceleration by <expression>.*
◯ *Valid only for the specified MOVE statement and interpolation movement.*

EXAMPLE:

```
M O V E  L , P 0 , A C C = 20
```

---------- *Moves from the current position to P0 at 20% acceleration.*

POINT

The ACC option cannot be used with Ver. 6.20 or earlier.

CAUTION

If the specified value is too small, the speed may not reach its maximum rate and a constant speed may not be maintained during continuous operation of the robot.

- *Deceleration setting*

FORMAT:

```
D E C = <expression>
```

The value of <expression> is 1 to 100.

EXPLANATION:

◯ *Specifies the ratio to the current robot deceleration by <expression>.*
◯ *Valid only for the specified MOVE statement and interpolation movement.*

EXAMPLE:

```
M O V E  L , P 0 , D E C = 10
```

---------- *Moves from the current position to P0 at 10% deceleration.*

POINT

The DEC option cannot be used with Ver. 6.20 or earlier.

CAUTION

If the specified value is too small, the speed may not reach its maximum rate and a constant speed may not be maintained during continuous operation of the robot.

EXAMPLES:

M O V E L , P 0 , A C C = 20

---------- Moves linearly from the current position to P0 at 20% acceleration.

M O V E L , P 1 , P 2 , P 3

---------- Moves linearly to P1, P2 and then P3 following the above movement.

M O V E L , P 4 , D E C = 80

---------- Moves linearly from the current position to P4 at 80% deceleration.

W A I T A R M

---------- Waits until the above movement is complete.

POINT

- *Except for a moving target axis, it is possible to move by multi task. When the 5th axis is set to an auxiliary axis, asynchronous movement is possible.*

```
Task 1:          :
          MOVE P, P0, P1, P2, S=100
                 :
Task 2:          :
          DRIVE(5, P3)
                 :
```

- *Linear and circular interpolation are only executed in Task 1 (Main task) and direct command.*
- *The circular interpolation radius is 5000.00mm maximum and 1.00mm minimum.*
- *Circular interpolation must be certainly specified with an even number.*

*Refer to axis parameter in "SYSTEM" mode for details on *1, *2 and *3.*

CAUTION

◯ ***PTP ultra-low speed movement***

When the movement pulse is set to (Encoder)*(Automatic speed)*(Program speed) within 20,000, the robot will move at the specified movement pulse for 5 minutes. (Movement pulse must be 118 pulse or more.)

EXAMPLE: With 4000pulse encoder speed 4%*1%

Minimum movement speed 0.39pulse/second (118/300)

Controllable speed resolution 0.00003p/second (1/300)

The speed absolute value and the resolution are very accurate, but the robot repeats only a "move and stop" in small amounts, in an unsatisfactory robot movement of poor quality. It is essential not to make any errors in the application.

◯ ***Circular interpolation***

When a small radius circular interpolation is performed with a heavy duty robot, the circular arc may be distorted. A similar phenomenon may occur with a light duty robot depending on the payload.

◯ ***Continuous interpolation***

When a linear interpolation and circular interpolation are continuously performed, the axis moves along a path slightly inner from the actual path at connecting point of line segments. The extent of this inner movement will be larger as the speed increases. If a continuous interpolation is performed between the line segments which are greatly different in distance, the axis speed may fluctuate.

RELATED COMMANDS: WAIT ARM, MOVEI, DRIVE, DRIVEI, SPEED, ASPEED

M O V E 2 Statements

FORMAT:

M O V E 2	*P T P* *P*	*, <point definition>[, <option> [, <option>, , ,]]*

EXPLANATION:

This statement executes moving commands for the sub robot. Therefore, only axes which are set as the sub robot axes can be moved with this command. Main robot axis and auxiliary axes cannot be moved.

Movement type : PTP
Point definition : Cartesian coordinates, point definition
Option : Speed setting, arch motion setting, STOPON conditions setting

Movement types

- *PTP (Point to Point)*
  *After confirming that a moving axis is placed in the tolerance*1 of current position, movement begins and then the command terminates when the arms enters the OUT effective position*2 and then execution of the next command starts.*
  Use WAIT ARM2 statements (described later) when waiting for positioning to within the tolerance area.

EXAMPLE:

```
M O V E 2  P T P ,  P 0
        ---------- Moves from current position to P0.
```

Point definition types

- *Cartesian coordinates*

FORMAT:

```
X  Y  Z  R  A  B
```

X, Y, Z, R, A, B are the coordinates values of each axis which is separated by spaces.

EXPLANATION:
Specifies the Cartesian coordinates.

EXAMPLE:

```
MOVE2  P,  1000  1000  1000  1000  0  0
          ----------------- Moves to the specified position by PTP control.
```

POINT

For Ver.5.09 and earlier, cannot use cartesian coordinates.

- *Point definition*

FORMAT:

```
<point expression>[, point expression>, •••, <point expression>]
```

EXPLANATION:
◯ *Sets a point number.*
◯ *Point definition may contain many points, separated by "," commas.*

EXAMPLE:

```
MOVE2  P,  P110+P111
          ----------------- Moves from the current position to the position
                            which is calculated by P110+P111.
```

Option types

- *Speed setting 1*

FORMAT:

S P E E D *S*	*= <expression>*

The value of <expression> is 1 to 100.

EXPLANATION:

◯ *The ratio for the robot (characteristic) maximum speed is set by <expression>.*

◯ *This setting is valid only for the specified MOVE2 statement.*

EXAMPLE:

```
M O V E 2   P ,   P 0 ,   P 1 ,   P 2 ,   S = 1 0
        ----------------- Moves at 10% speed in the sequence; P0, P1, P2.
```

- *Arch motion setting*

FORMAT:

x=<expression>[, x=<expression>, •••]

In the expression, x specifies the X, Y, Z, R axes.
When the <expression> is in real numbers, the units are in mm. When it is in integers, the units are in pulses.

EXPLANATION:

◯ *First, the X-axis moves to <expression> position, then other axes move to the target position, and finally the X-axis to the target position.*

◯ *The arch motion can only be specified for PTP movement.*

◯ *Execution of arch motion including the X or Y-axis is possible, when the target position or the value of <expression> is in integers in the SCARA type robot, and when the value of <expression> is integers in the XY type robot.*

◯ *The arch starting position can be changed by arch motion position *3.*

EXAMPLE:

```
M O V E 2   P ,   P 1 ,   Z = 0
        ----------------- First, the Z-axis moves to 0 pulse, other axes moves
                          to P1, and finally Z-axis moves to P1.
```

- *STOPON conditions setting*

FORMAT:

```
STOPON  <condition expression>
```

EXPLANATION:

◯ *Specify where you wish to stop when the <condition expression> are setup.*
When conditions are set during the start of movement, the next command is executed without movement.

◯ *When PTP or linear interpolation is set, the STOPON conditions setting can be used. In such a case, the next command is executed after entering within tolerance of the target position in linear interpolation.*

◯ *Only program execution is valid.*

EXAMPLE:

```
MOVE2 P, P100, S=70, STOPON DI(31)=1
```

---------- *Moves at 70% speed to P100. During movement to DI(31)=1, and then stops.*

EXAMPLE:

<Example 1>

```
FOR A=1 TO 10
    MOVE2 P, P[A]
NEXT A
```

---------- *Moves from current position in the sequence of; P1, P2, P3, •••, P10.*

<Example 2>

```
FOR A=1 TO 10
   DO(30)=0
   MOVE2 P, P0, Z=10000
   DO(30)=1
   DELAY 500
   MOVE2 P, P[A], Z=0
   DO(30)=0
   DELAY 500
NEXT A
```

---------- Moves from work feed position P0 to P1, P2, P3, •••, P10.

POINT

Except for a moving target axis, it is possible to move by multi task. When set to main/sub robot, asynchronous movement is possible.

```
Task 1:          :
          MOVE P, P0, P1, P2, S=100
                         :
Task 2:          :
          MOVE2 P, P0, P1, P2, S=100
                         :
```

*Refer to axis parameter in "SYSTEM" mode for details on *1, *2 and *3.*

RELATED COMMANDS: WAIT ARM2, MOVEI2, DRIVE2, DRIVEI2, SPEED2, ASPEED2

M O V E I Statements

FORMAT:

M O V E I	*P T P* *P*	*, <point definition>[, option[, option, , ,]]*

EXPLANATION:
This statement executes relative movement commands of the main robot. Therefore, only axes which are set as the main robot axes can be moved with this command. Sub robot axis and auxiliary axes cannot be moved.

Movement type : PTP
Point definition : Cartesian coordinates, point definition
Option : Speed setting

Movement types

- *PTP (Point to Point)*
  *After confirming that a moving axis is placed in the tolerance*1 of current position, movement begins and then the command terminates when the arms enters the OUT effective position*2 and then execution of the next command starts.*
  Use WAIT ARM statements (described later) when waiting for positioning to within the tolerance area.

  EXAMPLE:
  M O V E I P T P , P 0
  ---------- Moves from current position, by the point amount shown by P0.

Point definition types

- *Cartesian coordinates*

FORMAT:

```
X  Y  Z  R  A  B
```

X, Y, Z, R, A, B are the coordinates values of each axis which is separated by spaces.

EXPLANATION:
Specifies the Cartesian coordinates.

EXAMPLE:

```
MOVEI  P,  1000  1000  1000  1000  0  0
       ----------------- Moves by a specified point amount from current
                         position, while under PTP control.
```

POINT

For Ver.5.09 and earlier, cannot use cartesian coordinates.

- *Point definition*

FORMAT:

```
<point expression>[, point expression>, •••, <point expression>]
```

EXPLANATION:
◯ *Sets a point number.*
◯ *Point definition may contain many points, separated by "," commas.*

EXAMPLE:

```
MOVEI  P,  P110+P111
       ----------------- Moves from the current position to the position
                         which is calculated by P110+P111.
MOVEI  P,  P[A]
       ----------------- Moves from current position, by the specified point
                         amount shown in P[A].
```

Option types

- *Speed setting 1*

FORMAT:

S P E E D S	= <expression>

The value of <expression> is 1 to 100.

EXPLANATION:

◯ *The ratio for the robot (characteristic) maximum speed is set by <expression>.*

◯ *This setting is valid only for the specified MOVEI statement.*

EXAMPLE:

```
M O V E I  P ,  P 0 ,  P 1 ,  P 2 ,  S = 1 0
```

---------- *Moves at 10% speed by the point amount indicated in the sequence; P0, P1, P2.*

EXAMPLE:

```
F O R  A = 1  T O  1 0
    M O V E I  P ,  P [ A ]
N E X T  A
```

---------- *Moves from current position by the point amount which is indicated in the sequence; P1, P2, P3, •••, P10.*

POINT

Except for a moving target axis, it is possible to move by multi task. When the 5th axis is set to an auxiliary axis, asynchronous movement is possible.

```
Task 1:          :
          MOVEI P, P0, P1, P2, S=100
                           :
Task 2:          :
          DRIVE(5, P3)
                           :
```

*Refer to axis parameter in "SYSTEM" mode for details on *1 and *2.*

RELATED COMMANDS: WAIT ARM, MOVE, DRIVE, DRIVEI, SPEED, ASPEED

M O V E I 2 Statements

FORMAT:

M O V E I 2	*P T P* *P*	*, <point definition>[, option[, option, , ,]]*

EXPLANATION:
This statement executes relative movement commands of the sub robot. Therefore, only axes which are set as the sub robot axes can be moved with this command. Main robot axis and auxiliary axes cannot be moved.

Movement type : PTP
Point definition : Cartesian coordinates, point definition
Option : Speed setting

Movement types

- *PTP (Point to Point)*
  *After confirming that a moving axis is placed in the tolerance*1 of current position, movement begins and then the command terminates when the arms enters the OUT effective position*2 and then execution of the next command starts.*
  Use WAIT ARM2 statements (described later) when waiting for positioning to within the tolerance area.

  EXAMPLE:
  M O V E I 2 P T P , P 0
  ---------- Moves from current position, by the point amount shown by P0.

Point definition types

- *Cartesian coordinates*

FORMAT:

```
X  Y  Z  R  A  B
```

X, Y, Z, R, A, B are the coordinates values of each axis which is separated by spaces.

EXPLANATION:
Specifies the Cartesian coordinates.

EXAMPLE:

```
MOVEI2  P,  1000 1000 --------- 1000          1000 0     0
                ----------------- Moves to the specified point amount under PTP
                                  control.
```

POINT

For Ver.5.09 and earlier, cannot use cartesian coordinates.

- *Point definition*

FORMAT:

```
<point expression>[, point expression>, •••, <point expression>]
```

EXPLANATION:
◯ *Sets a point number.*
◯ *Point definition may contain many points, separated by "," commas.*

EXAMPLE:

```
MOVEI2  P,  P110+P111
                ----------------- Moves from the current position to the position
                                  which is calculated by P110+P111.
```

Option types

- *Speed setting 1*

FORMAT:

SPEED S	*= <expression>*

The value of <expression> is 1 to 100.

EXPLANATION:

◯ *The ratio for the robot (characteristic) maximum speed is set by <expression>.*

◯ *This setting is valid only for the specified MOVEI2 statement.*

EXAMPLE:

```
MOVEI2 P, P0, P1, P2, S=10
        ---------- Moves at 10% speed by the point amount which is
                    indicated in the sequence; P0, P1, P2.
```

EXAMPLE:

```
FOR A=1 TO 10
    MOVEI2 P, P[A]
NEXT A
        ---------- Moves from current position, by the point amount
                    indicated in the sequence; P1, P2, P3, •••, P10.
```

POINT

Except for a moving target axis, it is possible to move by multi task. When set to main/sub robot, asynchronous movement is possible.

```
Task 1:          :
          MOVEI P, P0, P1, P2, S=100
                         :
Task 2:          :
          MOVE2 P, P0, P1, P2, S=100
                         :
```

*Refer to axis parameter in "SYSTEM" mode for details on *1 and *2.*

RELATED COMMANDS: WAIT ARM2, MOVE2, DRIVE2, DRIVEI2, SPEED2, ASPEED2

ON ERROR GOTO Statements

FORMAT:

ON ERROR GOTO	<label> 0

Output Data:
Function ERR =Error Code
Function ERL =Line Number of Error

EXPLANATION:
Even if an error occurs, it is possible to avoid interrupting the program by having the robot jump to an error processing routine prepared at the location of the <label>. Then the program can continue.
The function error code and the line on which the error occurred are saved and these can be used in the error processing routine.
To return to the program from the error processing routine, use the RESUME statement.
If "ON ERROR GOTO 0" is executed, the error interruption becomes ineffective. If an error occurs, an error message will be displayed and the program will stop.
If "ON ERROR GOTO 0" is executed in the error processing routine, the appropriate error message is displayed and the program is stopped.
For error code meanings, refer to the separate "2 Error Messages" in Chapter 8 on the User's Manual.

EXAMPLE:

```
        ON ERROR GOTO *ER1
          ---------- If error occurs, jump to ER1.
        FOR A=0 TO 9
        B=A+10
        P[B]=P[A]
        NEXT A
        HALT
    *ER1:
        IF ERR=&H0604 THEN *NEXT L
          ---------- When error code is "&H0604" (no point), jump to
                     NEXT.
        ON ERROR GOTO 0
          ---------- Error message is displayed and program is stopped.
    *NEXT L:
        RESUME NEXT
          ---------- Jump to statement after error and continue program.
```

POINT

◯ *When "ON ERROR GOTO <label>" is used, the last one executed is effective.*

◯ *If an error occurs during the execution of the error processing routine, the robot will stop after displaying the error message. An error trap does not occur in an error routine.*

◯ *"ON ERROR GOTO <label>" may not be used in error processing routines.*

RELATED COMMAND: ERL, ERR, RESUME

ON and GOTO Statements , ON and GOSUB Statements

These statements may be used in the following ways:

FORMAT:

ON <expression>	GOTO GO TO	<label 1> [, <label 2>. . .]
ON <expression>	GOSUB GO SUB	<label 1> [, <label 2>. . .]

EXPLANATION:
The value of the <expression> is what determines where the robot will jump to next. When the value of the <expression> is "1", the robot will jump to the first <label 1>. When it is 2, it will jump to the second <label 2>. The value of the <expression> must be an integer, and it will determine which of the specified labels the robot will jump to (the robot will jump to the nth <label n>).
After executing the subroutine at the <label> specified by the ON/GOSUB statement, the robot will return to the command following the ON/GOSUB statement. If the value of the <expression> is 0 or greater than the number of labels specified, the robot will continue to the next command. If the value of the expression is negative, there will be an error message "Illegal function call", and the program will stop.

EXAMPLE:

```
*L0:
    ON DI3() GOSUB *L1, *L2, *L3
    GOTO *L0
*L1:
    MOVE P, P0
    RETURN
*L2:
    DO3(0)=1
    DELAY 1000
    RETURN
*L3:
    DO3(1)=1
    WAIT DI4(7)=1
    RETURN
```

RELATED COMMAND: DECLARE, GOSUB, RETURN

ONLINE and OFFLINE Statements

FORMAT:

```
ONLINE
OFFLINE
```

EXPLANATION:
This command selects the mode of communication. if ONLINE/OFFLINE is selected, the communication ports are initialized.
This command has the same result as changing the online/offline communication parameters of the "SYSTEM" mode.

EXAMPLE:

```
OFFLINE
SEND CMU TO N
SEND CMU TO P0
ONLINE
HALT
```

POINT

After the communication ports have been initialized, communication is possible once communication errors and the receiving buffer has been cleared.
For details regarding communication, refer to the separate "RS-232C Interface" in Chapter 6 on the User's Manual

O R G O R D Statements (Return to Origin Sequence Setting Statement for Main Group)

FORMAT:

```
ORGORD  <expression 1>
```

EXPLANATION:
This command statement sets the axis sequence parameter which performs return to origin movement of the main group.

<expression 1>: integer of 0 to 654321

EXAMPLE:

```
A = 3
O R G O R D  A
O R I G I N ---------  After completing return to origin movement of the
                       3rd axis of main group, for the remaining axis, per-
                       forms simultaneous return to origin movement.
M O V E  P , P 0
     :
H A L T
```

RELATED COMMAND: ABSRST, ORIGIN, ORGORD2

O R G O R D 2 Statements (Return to Origin Sequence Setting Statement for Sub Group)

FORMAT:

```
O R G O R D 2  <expression 1>
```

EXPLANATION:
This command statement sets the axis sequence parameter which performs return to origin movement of the sub group.

<expression 1>: integer of 0 to 4321

EXAMPLE:

```
A = 1 2 3 4
O R G O R D 2  A
O R I G I N --------- Return to origin movement for axes of the sub group
                      is performed in the sequence of 1, 2, 3, 4.
M O V E 2  P , P 0
      :
H A L T
```

RELATED COMMAND: ABSRST, ORIGIN, ORGORD

O R I G I N Statements

FORMAT:

O R I G I N

EXPLANATION:
This command statement executes incremental motor axis origin return for the robot.
A shutdown while movement is in-progress, will cause an incomplete return to origin.
When setting two robots, return to origin movement for the sub robot group is performed after completing return to origin movement for main group.

EXAMPLE:

O R I G I N --------- Performs return to origin movement of incremental motor.

RELATED COMMAND: ABSRST, ORGORD, ORGORD2, MCHREF, MCHREF2

OUTPOS Statements (Out Effective Position Setting for Main Group)

FORMAT 1:

```
OUTPOS <expression>
```

FORMAT 2:

```
OUTPOS (<expression 1>)=<expression 2>
```

The value of <expression 1> must be from 1 to 6 (axis number).

EXPLANATION:
This command statement changes the out effective position parameter for the main group to the value specified in <expression>. Format 1 changes all axes of the main group. Format 2 changes the out effective position parameter for the axis specified in <expression 1> to the value specified in <expression 2>.

POINT

If an axis is set when "no axis" has been specified in "GENERATE" mode, there will be an error message "Specification mismatch" to remind the user of the conflict in usage. The execution of the program will also be halted.

EXAMPLE:

```
'CYCLE WITH DECREASING OUTPOS
DIM SAV(3)
GOSUB *SAVE_OUTPOS
FOR A=1000- TO 1000 STEP 1000
    GOSUB *CHANGE_OUTPOS
    MOVE P,P0
    DO3(0)=1---------- Chuck closes
    MOVE P,P0
    DO3(0)=0---------- Chuck opens
NEXT A
GOSUB *RESTORE_OUTPOS
HALT
*CHANGE_OUTPOS:
    FOR B=1 TO 4
      OUTPOS(B)=A
    NEXT B
RETURN
*SAVE_OUTPOS:
    FOR B=1 TO 4
      SAV(B-1)=OUTPOS(B)
    NEXT B
RETURN
```

O U T P O S 2 Statements (Out Effective Position Setting for Sub Group)

FORMAT 1:

```
O U T P O S 2 <expression>
```

FORMAT 2:

```
O U T P O S 2 (<expression 1>)=<expression 2>
```

The value of <expression 1> must be from 1 to 4 (axis number).

EXPLANATION:
This command statement changes the out effective position parameter for the sub group to the value specified in <expression>. Format 1 changes all axes of the sub group. Format 2 changes the out effective position parameter for the axis specified in <expression 1> to the value specified in <expression 2>.

POINT

- *This command is valid only when the sub group has been set in system generation.*
- *If an axis is set when "no axis" has been specified in system generation mode, there will be an error message "Specification mismatch" to remind the user of the conflict in usage. The execution of the program will also be halted.*

EXAMPLE:

```
'CYCLE WITH DECREASING_OUTPOS
DIM SAV(3)
GOSUB *SAVE_OUTPOS
FOR A=1000- TO 1000 STEP 1000
    GOSUB *CHANGE_OUTPOS
    MOVE2 P,P0
    DO3(0)=1---------- Chuck closes
    MOVE2 P,P0
    DO3(0)=0---------- Chuck opens
NEXT A
GOSUB *RESTORE_OUTPOS
HALT
*CHANGE_OUTPOS:
    FOR B=1 TO 4
      OUTPOS2(B)=A
    NEXT B
RETURN
*SAVE_OUTPOS:
    FOR B=1 TO 4
      SAV(B-1)=OUTPOS2(B)
    NEXT B
RETURN
*RESTORE_OUTPOS:
    FOR B=1 TO 4
      OUTPOS2(B)=SAV(B-1)
    NEXT B
RETURN
```

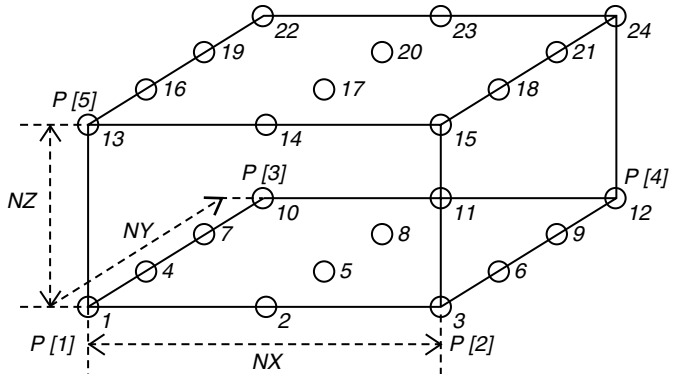
P D E F Statements

FORMAT:

PDEF(<expression 1>) =<expression 2>,<expression 3>,<expression 4>

EXPLANATION:

This command statements defines the palette used to execute the palette movement command.



<expression 1> : Palette definition number (0 to 9)
<expression 2> : Number of points between P[1] and P[2]
<expression 3> : Number of points between P[1] and P[3]
<expression 4> : Number of points between P[1] and P[5]

1 ≤ (<expression 2> × <expression 3> × <expression 4>) ≤ 32768

EXAMPLE:

P D E F (1) = 3 , 4 , 2
Palette definition 1 is defined as 3 × 4 × 2.

CAUTION

◯ ***Point data used in palette definition uses the data area as follows.***
QRC, QRCH, MRC (with extension RAM), MRCH Series
Palette definition 0: P[1]-P[5] P3996-P4000
Palette definition 1: P[1]-P[5] P3991-P3995
:
Palette definition 9: P[1]-P[5] P3951-P3956

MRC (without extension RAM) Series
Palette definition 0: P[1]-P[5] P1596-P1600
Palette definition 1: P[1]-P[5] P1591-P1595
:
Palette definition 9: P[1]-P[5] P1551-P1556

◯ ***This command cannot be used with versions prior to V5.21.***

P M O V E Statements

FORMAT:

```
P M O V E (<expression 1>,<expression 2>)[,option[,option · · · · ]]
```

EXPLANATION:
This command statement executes the palette moving command for the main robot. Only axes which are set as the main robot axes can be moved by this command. An auxiliary axis cannot be moved.
After arm positioning to within the effective tolerance position, movement begins and the command terminates when the arms enter the OUT effective position. Use the WAIT ARM statement for continuing on to another job operation after positioning within the tolerance area.

<expression 1> : The palette definition number (0 to 9)
<expression 2> : The point number on the palette (1 to 32768)

There are options in the PMOVE statement as shown below.
Arch movement setting definitions : First of all, the specified axis moves to the specified position, next, other axes which are not specified move to the target position, and finally, the specified axis moves to the target position.

```
x=<expression 3>      (x=X, Y, Z, R, A, B)
```

Speed specification : The speed settings are made in units of %. It is only enabled when a command is specified.

```
| SPEED | =<expression 4>
| S     |
```

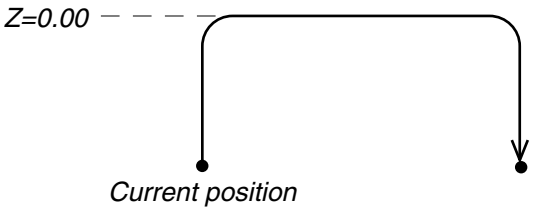
STOPON condition specification : The STOPON condition defines conditions under which the arm will stop during movement. This STOPON setting is enabled until the arm enters the out effective position.

```
STOPON <DI/DO conditional expression>
```

EXAMPLE:

PMOVE (1,3),Z=0.00

------Move from current position to the third point of palette definition 1.



PMOVE (1,10),S=10

------Move from current position to the 10th point of palette definition 1 at 10% speed.

PMOVE (N,M),STOPON DI(21)=1

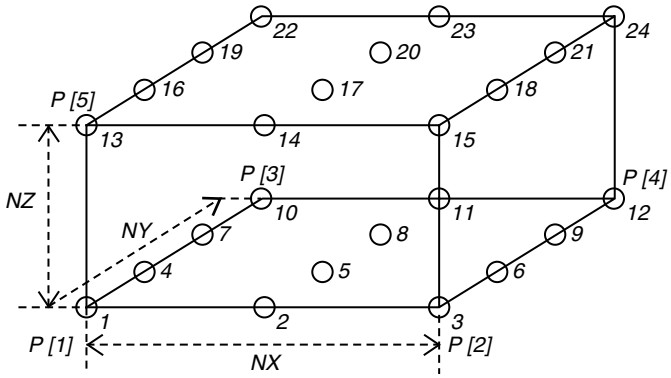
------Move to Mth point of palette definition N. Meanwhile, if all-purpose input 21 is turned on, it stops at that position.

POINT

This command cannot be used with versions prior to V5.21.

CAUTION

Palette movement will not function correctly unless point data is input in the sequence as follows.



When the main robot moves with the PMOVE statement, the R axis moves to the position specified by P[1].

P M O V E 2 Statements

FORMAT:

```
P M O V E 2 (<expression 1>,<expression 2>)[,option[,option · · · ·]]
```

EXPLANATION:
This command statement executes the palette moving command for the sub robot. Only axes which are set as the main robot axes can be moved with this command. An auxiliary axis cannot be moved.
After arm positioning is complete, (to within effective position tolerance), movement begins and the command terminates when the arms enter the out effective position. Use the WAIT ARM2 statement for continuing on to another job operation after positioning to within the tolerance area.

<expression 1> : The palette definition number (0 to 9)
<expression 2> : The point number on the palette (1 to 32768)

There are options in PMOVE2 statement as shown below.

Arch movement setting definitions : First of all, the specified axis moves to the specified position, next, other axes which are not specified move to the target position, and finally, the specified axis moves to target position.

```
x=<expression 3>      (x=X, Y, Z, R)
```

Speed specification : The speed settings are made in units of %. It is only enabled when a command is specified.

```
| SPEED | =<expression 4>
| S     |
```

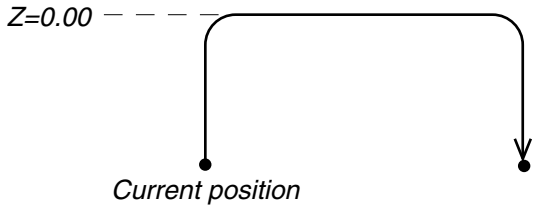
STOPON condition specification : The STOPON condition defines conditions under which the robot will stop during movement.
This STOPON setting is valid until entering out effective position.

```
STOPON <DI/DO conditional expression>
```

EXAMPLE:

P M O V E 2 (1 , 3) , Z = 0 . 0 0

------Move from current position to the third point of the palette definition 1.



P M O V E 2 (1 , 1 0) , S = 1 0

------Move from present position to the 10th point of palette definition at 10% speed.

P M O V E 2 (N , M) , S T O P O N D I (2 1) = 1

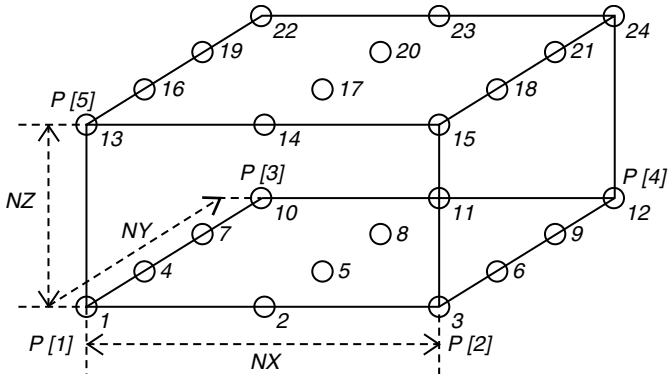
------Move to Mth point of palette definition N. Meanwhile, if all-purpose input 21 is turned on, it stops at that position.

POINT

This command cannot be used with versions prior to V5.21.

CAUTION

Palette movement will not function correctly unless point data is input in the sequence as follows.



When the sub robot moves with the PMOVE2 statement, the R axis moves to the position specified by P[1].

P R I N T Statements

This command displays data on the MPB screen.

FORMAT:

P R I N T [<expression>] [, \| ;] <expression>...] [, \| ;]

EXPLANATION:

- *If a value or character string is put in the place of the <expression>, that value or character string will be sent to the screen as it is. Variables and character variables will result in the value or character string assigned to them being sent to the screen. If no <expression> is specified, a carriage return will be output to the screen.*
- *Numbers that are correctly expressed as integers are output in a format without any decimals.*
- *If the length of the data to be output to the screen exceeds the length of the line on the screen, a carriage return will be inserted and the remainder of the data will appear on the following line of the screen.*
- *The comma may be used in the following ways:*
  a. *When the present position is the right end, a carriage return is sent to the screen.*
  b. *When the data is too long for the line, a carriage return is sent to the screen and the data is displayed on the following line.*
  c. *In cases other than the above, a space is sent to the screen, and the data is displayed on the screen in the next printing column.*
  d. *Each printing column is 12 characters wide.*
- *If the semicolon is used, the ensuing data will be displayed immediately after the last data displayed on the screen. However, the data will be as in a. and b.. In the case of a character string, all characters that fit on the same line are displayed there, with any remaining characters displayed on the following line.*
- *PRINT statements with no delineators (commas, semicolons) will send a carriage return to the screen automatically and the end of each line.*
- *PRINT statements that do end with delineators print following PRINT statements on the same line.*

EXAMPLES:

```
PRINT "COLUMN=";X, "LINE=";Y
PRINT A%, B!, C$
```

RELATED COMMAND: INPUT, SEND

P n (Point Definition Statements)

This command is used to define points within user programs.

FORMAT:

```
P n = X  Y  Z  R  A  B
```

n :	*Point number*	*Maximum 4 digits*
X:	*X-axis coordinate*	*Maximum 6 digits*
Y:	*Y-axis coordinate*	*Maximum 6 digits*
Z:	*Z-axis coordinate*	*Maximum 6 digits*
R:	*R-axis coordinate*	*Maximum 6 digits*
A:	*A-axis coordinate*	*Maximum 6 digits*
B:	*B-axis coordinate*	*Maximum 6 digits*

EXPLANATION:
This statement defines a point in the movement range of the robot. It is used when changing point data while a user program is being executed.

POINT

◯ *Coordinates must be numerals.*
◯ *X, Y, Z, R, A, B axis data must be separated with a space.*

EXAMPLE:

```
P1=    100000 -200000      0      0      0      0
P123=100.00 -200.00 80.00  0.00   0.00   0.00
MOVE P, P1
MOVE P, P123
HALT
```

CAUTION

When the power is turned off during execution of a point definition statement, it may cause a "9.2: Point data destroyed" message to be issued.
This is issued when the CPU halts processing before rewriting of point data is complete.
In programs where a particularly high frequency of point definition statement repetitions occur, this error message may be issued frequently.

RELATED COMMAND: Point assignment statement, Point element assign-ment statement, LOCx

R E M (Comments)

FORMAT:

R E M *'*	*<character string>*

EXPLANATION:
All characters that follow REM or an apostrophe (') are ignored by the robot. These can be used to make comments throughout the program for easy reference.

EXAMPLES:

```
R E M  *** M A I N  P R O G R A M ***
'  *** S U B R O U T I N E  ***
```

R E S E T Statements

FORMAT:

R E S E T	*D O m ([b, • • • , b])* *D O (m b, • • • , m b)* *M O n ([b, • • • , b])* *M O (n b, • • • , n b)* *T O 0 ([b, • • • , b])* *T O (0 b, • • • , 0 b)* *L O 0 ([b, • • • , b])* *L O (0 b, • • • , 0 b)*

m: Port number 2 to 7, 10 to 11
n: Port number 2 to 7, 10 to 13
b: Bit definition 0 to 7

EXPLANATION:
This turns OFF the bits output through the DO, MO, TO and LO ports. However, DO(27) cannot be used for the QRCH-E controller as it is already used as a custom output.

POINT

◯ *Output is not possible to the DO0, DO1, MO0 and MO1 ports.*
◯ *Bit should be defined in ascending order from the right.*

EXAMPLES:

R E S E T D O 2 ( )
---------- DO(20) to DO(27) are OFF.

R E S E T D O 1 (7 , 3 , 0)
---------- DO(17), (13), (10) are OFF.

R E S E T M O (4 1 , 3 0 , 2 7)
---------- MO(41), (30), (27) are OFF.

R E S E T L O 0 (3 , 2)
---------- LO(03) and LO(02) are OFF.

RELATED COMMAND: DO, LO, MO, SET, TO

RESTART Statements

FORMAT:

```
RESTART Tn
```

n = 2 to 8

EXPLANATION:
This command restarts a task after a temporary stop.

POINT

To use this command, a high-speed arithmetic processor (option) must be provided.

EXAMPLES:

```
        START *SUBTASK, T2
        FLAG=1
    *L0:
        IF FLAG=1 AND DI2(0)=1 THEN
          SUSPEND T2
          FLAG=2
          WAIT DI2(0)=0
        ENDIF
        IF FLAG=2 AND DI2(0)=1 THEN
          RESTART T2
          FLAG=1
          WAIT DI2(1)=0
        ENDIF
        MOVE P, P0
        MOVE P, P1
        GOTO *L0
    *SUBTASK:
        DO2(0)=1
        DELAY 1000
        DO2(0)=0
        DELAY 1000
        GOTO *SUBTASK
```

RELATED COMMAND: CUT, EXIT TASK, START, SUSPEND

R E S U M E Statements

FORMAT:

R E S U M E	*[*	*NEXT* *<label>*	*]*

EXPLANATION:

After removing the cause of the error, restart program execution.
There are three ways to restart a program, depending on the place the program is started.

1. *RESUME* — *The program is restarted with the command that caused an error.*
2. *RESUME NEXT* — *The program is restarted with the command that follows the command that caused an error.*
3. *RESUME <label>* — *The program is restarted with the command fol lowing the <label> specified.*

EXAMPLE:

See section on O N E R R O R G O T O Statements.

POINT

It is possible to use RESUME statements in an error processing routine only.

RELATED COMMAND: ON ERROR GOTO

R I G H T Y and L E F T Y Statements

FORMAT:

```
R I G H T Y
L E F T Y
```

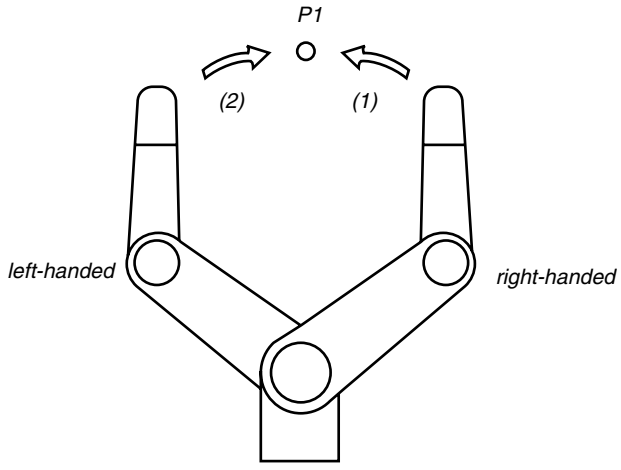
EXPLANATION:
This command determines whether the main robot will be "right-handed" or "left-handed" on a Cartesian coordinate system. The arm will not move merely upon making a selection.

POINT

This command is effective during program execution for SCARA type robots, for the MOVE and MOVEI commands. It may not be used with Cartesian robots.

EXAMPLE:

```
R I G H T Y
M O V E  P , P 1
L E F T Y
M O V E  P , P 1
R I G H T Y
H A L T
```



SCARA type robot

R I G H T Y 2 and L E F T Y 2 Statements

FORMAT:

```
R I G H T Y 2
L E F T Y 2
```

EXPLANATION:

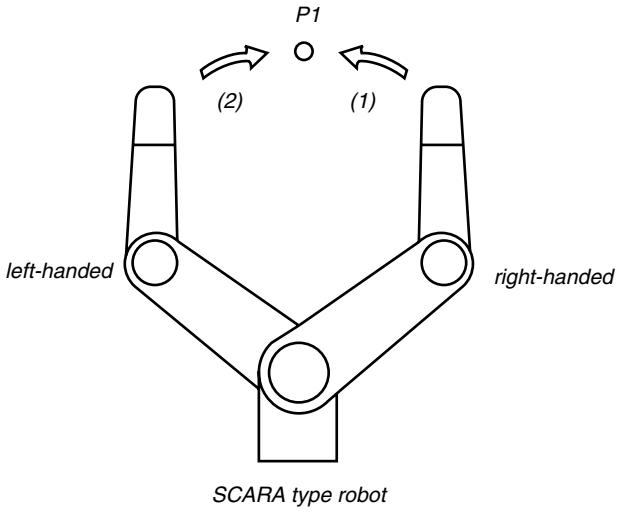
This command determines whether the sub robot will be "right-handed" or "left-handed" on a Cartesian coordinate system. The arm will not move merely upon making a selection.

POINT

- *This command is valid only when the sub group has been set in system generation.*
- *This command is effective during program execution for SCARA type robots, for the MOVE2 and MOVEI2 commands. It may not be used with Cartesian robots.*

EXAMPLE:

```
R I G H T Y 2
M O V E 2  P ,  P 1
L E F T Y 2
M O V E 2  P ,  P 1
R I G H T Y 2
H A L T
```



SCARA type robot

S n (Shift Coordinate Definition Statement)

FORMAT:

```
S m = X  Y  Z  R
```

m: Shift number (0 to 9)
X: X-axis shift amount *Unit: mm*
Y: Y-axis shift amount *Unit: mm*
Z: Z-axis shift amount *Unit: mm*
R: Revolution shift amount *Unit: fl*

EXPLANATION:
This command defines the shift coordinates in a user program. It may be used to change shift coordinates in a user program.

NOTE

- *Coordinates must be numerals.*
- *X, Y, Z, R data must be separated with spaces.*

EXAMPLE:

```
S0=  0.00    0.00-------- 0.00          0.00
S1=  100.00100.00--- 100.00       90.00
SHIFT S0
MOVE P, P0
MOVE P, P1
SHIFT S1
MOVE P, P0
MOVE P, P1
HALT
```

CAUTION

When the power is turned off during execution of a shift coordinate definition statement, it may cause a "9.6: Shift data destroyed" message to be issued.

RELATED COMMAND: Shift assignment statement, Shift element assignment statement, LOCx, SHIFT

S E L E C T C A S E Statements

This command executes any of multiple blocks.

FORMAT:

```
S E L E C T [CASE] <expression>
C A S E <expression list 1>
[Block 1]
[CASE<expression list 2>
[Block 2]]
     :
     :
[CASE ELSE
[Block n]]
E N D  S E L E C T
```

The CASE <expression list> is arranged with multiple numerical or characters punctuated with commas.

EXPLANATION:
The value of the SELECT statement <expression> determines which block between SELECT~END SELECT will be executed.
Block 1 is executed when the <expression> value of is equivalent to one of the <expression> values from the first case statement of <expression list 1>.
Block 2 is executed when the <expression> value of is equivalent to one of the <expression> values from the second case statement of <expression list 2>.
Execution is performed in the same way according to the <expression> values shown below.
When the <expression> value is not equivalent to all the values for the case statement <expression list>, the next line of block n of the CASE ELSE statement is executed.

EXAMPLE:

```
WHILE -1
SELECT CASE DI3()
CASE 1
CALL *EXEC(1,10)--------executed when DI3() is 1.
CASE 2
CALL *EXEC(21,30)-----executed when DI3() is 2.
CASE 3
CALL *EXEC(41,50)-----executed when DI3() is 3.
CASE ELSE ----------executed when DI3() is other than
                                    1, 2 or 3.
END SELECT
WEND
SUB *EXEC(A,B)
 FOR C=A TO B
  MOVE P, P0
  MOVE P, P[A]
 NEXT
END SUB
```

S E N D Statements

FORMAT:

[label:] S E N D <source file> T O <destination file>

EXPLANATION:
This command sends data from one file to a different file.
Data is sent from the <source file> to the <destination file>.

File FORMAT

Types	File name	Definition format		Read	Write
		All	Separate file		
User memory	System	ALL	—	OK	OK
	Program	PGM	<bbbbbbbb>	OK	OK
	Point Data	PNT	P"["x"]", Pn	OK	OK
	Parameter	PRM	—	OK	OK
	Shift Data	SFT	S"["x"]", Sn	OK	OK
	Hand Data	HND	Hn	OK	OK
	Palette definition Data	PLT	PLn	OK	OK
Variable,	Variable	VAR	ab...by	OK	OK
Numeral	Array Variable	ARY	ab...by(x)	OK	OK
	Numeral	—	"cccccc"	OK	NG
Status	Directory	DIR	<<bbbbbbbb>>	OK	NG
	Memory Amount	MEM	—	OK	NG
	Point Usage Status	SPN	—	OK	NG
	Machine Reference	MRF	—	OK	NG
Device	DI+DO+MO+LO+TO	DIO	—	OK	NG
	DI	—	DIn( )	OK	NG
	DO	—	DOn( )	OK	OK
	MO	—	MOn( )	OK	OK
	LO	—	LOn( )	OK	OK
	TO	—	TOn( )	OK	OK
	Communication	CMU	—	OK	OK
	Console Input	KEY	—	OK	NG
	Console Output	SCR	—	NG	OK
Others	File End Code	EOF	—	OK	NG

n: Numeral
a: Alphabetic character
b: Alphanumeric character or underscore (_)
c: Alphanumeric character or symbol
"[" "]" mean characters that are specified directly.
x: Statement
y: Format

POINT

Refer to "15 Data File Details" for more details.

EXAMPLES:

S E N D P G M T O C M U

---------- All programs will be sent to the communication port.

S E N D C M U T O P N T

---------- Communication port data will be assigned to point values.

S E N D P R M T O C M U

---------- All parameters will be sent to the communication port.

*S E N D " * * O K * * * " T O C M U*

*---------- Character string "**OK***" will be sent to communication port.*

S E N D P 1 0 0 T O S C R

---------- Point data for P100 will be sent to the screen (MPB).

S E N D < < P R O G 1 > > T O C M U

---------- Directory of program "PROG1"will be sent to the communication port.

S E N D D O 3 ( ) T O C M U

---------- DO port No. 3 status will be sent to the communication port.

S E N D E O F T O C M U

---------- End of file code (=1AH) will be sent to the communication port.

POINT

◯ *The SEND command may not be used to write into read-only files (D I R , M E M etc).*

Incorrect Examples:

S E N D C M U T O D I R

S E N D S C R T O P G M

◯ *Even if the files are used correctly, if the data format is not identical, the operation will not be performed with the desired results.*

Incorrect Examples:

S E N D P G M T O P R M

S E N D D I 1 ( ) T O S F T

RELATED COMMAND: INPUT, PRINT

S E R V O Statements

FORMAT:

S E R V O	*O N* *O F F* *F R E E* *PWR*	*[(<expression>)]*	

The value of <expression> must be from 1 to 6 (axis number).

EXPLANATION:
This controls the ON/OFF for servos of the axes (in the main group) specified axes number or all axes (in the main group and the sub group).
When the <expression> is not specified, control is set for all axes and the motor power supply simultaneously turns ON and OFF.

ON----- Turns the servo ON. If no axis is specified the motor power supply sets to ON.
OFF ---- Turns the servo OFF and the regenerative brake is applied. The brake is applied to the axis and locked. If no axis is specified the motor power supply also turns OFF.
FREE --- Turns the servo OFF and the regenerative brake is released. The brake is released from the axis. If no axis is specified the motor power supply also turns OFF.
PWR --- Turns ON the motor power supply.

CAUTION

When the motor power supply is OFF, servos for separate axes cannot be turned ON.

EXAMPLES:

S E R V O O N---------- Turns ON the servo for all axes when the motor power supply is set to ON.
S E R V O O F F ---------- Turns OFF the servo for all axes when the motor power supply is set to OFF. Brakes are applied and locked to axes having brakes.
S E R V O F R E E (3)---------- Turns OFF the servo for the 3rd axis (Z-axis) and releases the brake.

POINT

This command is executed after positioning (to within effective position tolerance) for arms of all axes (in the main group and the sub group).

S E R V O 2 Statements

FORMAT:

S E R V O 2	*O N* *O F F* *F R E E* *PWR*	*[(<expression>)]*	

The value of <expression> must be from 1 to 4 (axis number).

EXPLANATION:
This controls the ON/OFF for servos of the axes (in the sub group) specified axes number or of all axes (in the main group and the sub group).
When the <expression> is not specified, control is set for all axes and the motor power supply simultaneously turns ON and OFF.

ON----- Turns the servo ON. If no axis is specified the motor power supply sets to ON.
OFF ---- Turns the servo OFF and the regenerative brake is applied. The brake is applied to the axis and locked. If no axis is specified the motor power supply turns OFF.
FREE --- Turns the servo OFF and the regenerative brake is released. The brake is released from the axis. If no axis is specified the motor power supply turns OFF.
PWR --- Turns ON the motor power supply.

CAUTION

When the motor power supply is OFF, servos for separate axes cannot be turned ON.

EXAMPLES:

S E R V O 2 O N ---------- Turns ON the servo for all axes when the motor power supply is set to ON.
S E R V O 2 O F F---------- Turns OFF the servo for all axes when the motor power supply is set to OFF. Brakes are applied and locked to axes having brakes.
S E R V O 2 F R E E (3) ------- Turns OFF the servo for the 3rd axis (Z-axis) and releases the brake.

POINT

- *This command is valid only when the sub group has been set in system generation.*
- *This command is executed after positioning (to within effective position tolerance) for arms of all axes (in the main group and the sub group).*

S E T Statements

FORMAT:

S E T	D O m ([b, • • •, b])	[, <expression>]
	D O (m b, • • •, m b)	
	M O n ([b, • • •, b])	
	M O (n b, • • •, n b)	
	T O 0 ([b, • • •, b])	
	T O (0 b, • • •, 0 b)	
	L O 0 ([b, • • •, b])	
	L O (0 b, • • •, 0 b)	

m: Port number 2 to 7, 10 to 11
n: Port number 2 to 7, 10 to 13
b: Bit definition 0 to 7

EXPLANATION:
This command turns on DO, MO, LO or TO output bits.
The <expression> sets the pulse output time (unit: ms).
However, DO(27) cannot be used for the QRCH-E controller as it is already used as a custom output.

POINT

◯ *Output is not possible to DO0, DO1, MO0 and MO1.*
◯ *Bit should be defined in ascending order from the right.*

EXAMPLES:

```
S E T  D O 2 ( )
       ----------------- DO(20) to DO(27) are ON.

S E T  D O ( 4 1 ,  3 0 ,  2 7 )
       ----------------- DO(41), (30), (27) are ON.

S E T  M O 2 ( 7 ,  3 ,  0 ) ,  1 0 0
       ----------------- MO(27), (23), (20) are ON for 100ms.

S E T  T O 0 ( )
       ----------------- TO(00) to TO(07) are ON.
```

RELATED COMMAND: DO, LO, MO, RESET, TO

S H A R E D Statements

FORMAT:

```
S H A R E D  <simple variable> [ ( ) ] [, <simple variable> [ ( ) ]. . .]
```

EXPLANATION:

This command causes variables declared at the program code level not to be passed on as parameters, and makes it possible to refer to them with a sub-procedure.

The <simple variable> designates the program level variable in which the sub-procedure is used. Either a variable name or an array name (in the case of the parentheses) is specified. If an array is specified, the entire array is effected.

POINT

- *SHARED statements allow variables to be shared only between the program level code and procedure within the same program.*
- *The program level code is a program written outside of a sub-procedure.*

EXAMPLE:

```
D I M  Y ! ( 1 0 )
X ! = 2 . 5
Y ! ( 1 0 ) = 1 . 2
C A L L  * D I S T A N C E
C A L L  * A R E A
H A L T
S U B  * D I S T A N C E
    S H A R E D  X ! ,  Y ! ( )
    P R I N T  X ! ^ 2 + Y ! ( 1 0 ) ^ 2
       ---------- Variable is shared (global).
E N D  S U B
S U B  * A R E A
    D I M  Y ! ( 1 0 )
    P R I N T  X ! * Y ! ( 1 0 )
       ---------- Variable is not shared (local).
E N D  S U B
```

RELATED COMMAND: SUB, END SUB

S H I F T Statements (Shift Coordinate Setting Statement for Main Robot)

FORMAT:

```
S H I F T <shift variable>
```

EXPLANATION:
The shift data specified by the <shift variable> becomes the shift coordinate setting for the main robot.

EXAMPLE:

```
S H I F T  S 1
M O V E  P ,  P 1 0
S H I F T  S [ A ]
M O V E  P ,  P 2 0
H A L T
```

RELATED COMMAND: Shift definition statement, Shift assignment state-ment, Shift element assignment statement

S H I F T 2 Statements (Shift Coordinate Setting Statement for Sub Robot)

FORMAT:

```
S H I F T 2     <shift variable>
```

EXPLANATION:
The shift data specified by the <shift variable> becomes the shift coordinate setting for the sub robot.

EXAMPLE:

```
S H I F T 2  S 1
M O V E 2  P ,  P 1 0
S H I F T 2  S [ A ]
M O V E 2  P ,  P 2 0
H A L T
```

POINT

This command is valid only when the sub robot has been set in system generation.

RELATED COMMAND: Shift definition statement, Shift assignment state-ment, Shift element assignment statement

S P E E D Statements (Speed Setting Statement for Main Group)

FORMAT:

```
S P E E D<expression>
```

The value of <expression> must be from 1 to 100. (Unit: %)

EXPLANATION:
This command changes the moving command speed of the main groups after this statement, to the value specified by the <expression>.

POINT

- *Changes all the speeds for robot configuration axes and auxiliary axes.*
- *The operating speed is set to the product of the automatic moving speed which is set by MPB operation or ASPEED commands and the speed which is specified by the SPEED command in the program.*
  EXAMPLE: When the automatic moving speed is 80% and the speed set by the SPEED command is set to 50% then:
  *Moving speed =80%*50%=40%.*

EXAMPLE:

```
A S P E E D  1 0 0
S P E E D  7 0
M O V E  P ,  P 0
        ----------------- Move at 70%(=100*70) of speed from current position to P0.
S P E E D  5 0
M O V E  P ,  P 1
        ----------------- Move at 50%(=100*50) of speed from current position to P1.
M O V E  P ,  P 2 ,  S = 1 0
        ----------------- Move at 10%(=100*10) of speed from current position to P1.
```

RELATED COMMAND: ASPEED, ASPEED2, SPEED2

SPEED2 Statements (Speed Setting Statement for Sub Group)

FORMAT:

```
SPEED2    <expression>
```

The value of <expression> must be from 1 to 100. (Unit: %)

EXPLANATION:
This command changes the moving command speed of the sub groups after this statement, to the value specified by the <expression>.

POINT

- *Changes all the speeds for robot configuration axes and auxiliary axes.*
- *The operating speed is set to the product of the automatic moving speed which is set by MPB operation or ASPEED2 commands and the speed which is specified by the SPEED2 command in the program.*
  EXAMPLE: When the automatic moving speed is 80% and the speed set by the SPEED2 command is set to 50% then:
  *Moving speed =80%*50%=40%.*

EXAMPLE:

```
ASPEED2  50
SPEED2  70
MOVE2  P,  P0
        ---------- Move at 35%(=50*70) of speed from current position to P0.
SPEED2  50
MOVE2  P,  P1
        ---------- Move at 25%(=50*50) of speed from current position to P1.
MOVE2  P,  P2,  S=10
        ---------- Move at 5%(=50*10) of speed from current position to P1.
```

CAUTION

This command is valid only when the sub group has been set in system generation.

RELATED COMMAND: ASPEED, ASPEED2, SPEED2

S T A R T Statements

FORMAT:

```
S T A R T<label>, T n
```

n = 2 to 8

EXPLANATION:
This command designates the task beginning with <label> as task "n" and begins to execute it.

POINT

To use this command, a high-speed arithmetic processor (option) must be provided.

EXAMPLE:

```
        S T A R T  * S U B T A S K ,  T 2
           ---------- "*SUBTASK" is designated as task 2 and begun.
        A = 0
    * L 1 :
        M O V E  P ,  P 0
        M O V E  P ,  P 1
        G O T O  * L 1
    * S U B T A S K :
        W A I T  D I 2 ( 0 ) = 1
        A = A + 1
        P R I N  T  " C O U N T = " ; A
        W A I T  DI 2 ( 0 ) = 0
        G O T O  * S U B T A S K
```

RELATED COMMAND: CUT, EXIT TASK, RESTART, SUSPEND

S U B and E N D S U B Statements

FORMAT:

```
S U B   <label> [ ( <parameter> [, <parameter> . . . ] ) ]
    :
E N D   S U B
```

EXPLANATION:
These statements define sub-procedures.
The sub-procedure begins with the SUB declaration and ends with the END SUB declaration.
The <label> can be the name of any sub-procedure.
Blocks (collections of statements) conventionally defined by GOSUB and RETURN statements are referred to as "subroutines". Blocks defined by SUB and END SUB statements are referred to as sub-procedures.

All variables used with GOSUB are global variables. But all variables in sub-procedures are local variables by default. These variables can only be referred to within their range of definition within the sub-procedure. When referring to a global variable, the SHARED statement is used.

It is possible to call the sub-procedure many times in the same program. Each time the sub-procedure is called it is possible to give it a different set of variables. To pass the set of variables on use the <parameter>.

The <parameter> is composed of one of the following. Each is separated by a comma.

EXAMPLE:
- *Effective variable names: A%, B!, C$*
- *Array names (with parentheses, all the array is passed on): A()*

Local variable values are initialized with 0 or a null string whenever the sub-procedure is called.

When the END SUB statement is found in the program, the sub-procedure is ended and the robot returns to the line succeeding the line in which the sub-procedure was called.

POINT

◯ *All except the following statements may be used to define a sub-procedure:*

•SUB. . .END SUB

It is not allowed to nest sub-procedures.

However, it is allowed to call another sub-procedure from a sub-procedure.

•DECLARE

◯ *It is not allowed to use the name and call or return a value within a sub-procedure.*

EXAMPLE:

*PRINT *COMPARE(X!, Y!) Not possible*

◯ *A label can be used in a sub-procedure definition but in statements like GOTO and GOSUB, jumping cannot be done on sub-procedure outer labels.*

EXAMPLES:

In the example below, the variables in the sub-procedure "TEST" are local for "TEST". The program level code (programs written outside of the sub-procedure) variables are irrelevant.

```
I = 1
CALL *TEST
PRINT I
HALT
SUB *TEST
    I = 50
END SUB
```

In the example below, the sub-procedure "COMPARE" is called twice, and each time a different set of variables is passed to it.

```
X% = 4
Y% = 5
CALL *COMPARE(REF X%, REF Y%)
PRINT X%, Y%
Z% = 7
W% = 2
CALL *COMPARE(REF Z%, REF W%)
PRINT Z%, W%
HALT
SUB *COMPARE(A%, B%)
    IF A%<B% THEN
        TEMP% = A%
        A% = B%
        B% = TEMP%
   ENDIF
END SUB
```

RELATED COMMAND: CALL, DECLARE, EXIT SUB, SHARED

SUSPEND Statements

FORMAT:

```
SUSPEND Tn
```

n = 2 to 8

EXPLANATION:
This command is used to temporarily suspend the execution of a task.

POINT

To use this command, a high-speed arithmetic processor (option) must be provided.

EXAMPLE:

```
        START *SUBTASK, T2
        SUSFLG=0
*L0:
        MOVE P, P0
        MOVE P, P1
        WAIT SUSFLG=1
        SUSPEND T2
        SUSFLG=0
        GOTO *L0
*SUBTASK:
        WAIT SUSFLG=0
        DO2(0)=1
        DELAY 1000
        DO2(0)=0
        DELAY 1000
        SUSFLG=1
        GOTO *SUBTASK
```

RELATED COMMAND: CUT, EXIT TASK, RESTART

S W I Statements

This statement switches the current program.

FORMAT:

```
S W I  <program name>
```

EXPLANATION:
This statement switches from the current program to the specified program, starting from the No.1 line after executing compiling. The DO output and MO output status do not change while switching programs. However the variables and array variables are cleared. If an error occurs in compiling, the operation stops. A parenthesis "<", ">" is attached before and after the program name.

EXAMPLE:

```
S W I <A B C>
        ----------------- The program for execution switches to <ABC>.
```

POINT

- *When there is no program to switch with, the message "3.3:Program doesn't exist" is displayed and operation stops.*
- *When an error occurs during compiling, an error message is displayed and the program stops.*
- *The SWI command can only be executed within Task 1 (main task). When using within Tasks 2 through 8, the message "6.1:Illegal command" is displayed and operation stops.*
- *The key will not function while compiling.*
- *Compiling is always performed when the SWI statement is executed. This compile time differs depending on the program size to be switched.*

T O Statements (Timer)

FORMAT:

[L E T]	*T O m ([b , • • •, b])* *T O (m b , • • •, m b)*	*=<expression>*

m: Port numbers 0
b: Bit definition 0 to 7

EXPLANATION:
The specified value is output to TO. When the arm is in motion, the output waits until movement has been completed (the arm reaches the OUT effective position).

POINT

Be sure to define bits in ascending order from the right.
The timer function is not present in the robot program and can be used as an internal output like MO.

EXAMPLES:

T O ( ) = & B 1 0 1 1 1 0 0 0
---------- TO(03), (04), (05) and (07) are ON, TO(00), (01), (02) and (06) are OFF.

T O 0 (6 , 5 , 1) = & B 0 1 0
---------- TO(05) is ON and TO(01) and (06) are OFF.

T O (0 7 , 0 5 , 0 2 , 0 0) = A
---------- The 4 lower bits contents of the variable A are output to TO(07), (05), (02) and (01).

RELATED COMMAND: RESET, SET

TOLE Statements (Tolerance Setting Statement for Main Group)

FORMAT 1:

```
TOLE <expression>
```

FORMAT 2:

```
TOLE ( <expression 1> )=<expression 2>
```

The value of <expression> must be from 1 to 6 (axis number).

EXPLANATION:
This command statement changes the tolerance parameter for the main group to the value specified in <expression>. Format 1 changes all axes of the main group. Format 2 changes the tolerance parameter for the main group axis specified in <expression 1> to the value specified in <expression 2>, after moving the arm specified in <expression 1> to within the tolerance range.

POINT
If an axis is set when "no axis" has been specified in "GENERATE" mode, there will be an error message "Specification mismatch" to remind the user of the conflict in usage. The execution of the program will also be halted.

EXAMPLE:

```
        'CYCLE WITH DECREASING TOLERANCE
        FOR A=100 TO 20 STEP -20
        GOSUB *CHANGE_TOLE
        MOVE P, P0
        MOVE P, P1
        NEXT A
        HALT
    *CHANGE_TOLE:
        FOR B=1 TO 4
        TOLE(B)=A
        NEXT B
        RETURN
```

TOLE2 Statements (Tolerance Setting Statement for Sub Group)

FORMAT 1:

```
TOLE2 <expression>
```

FORMAT 2:

```
TOLE2 ( <expression 1> )=<expression 2>
```

The value of <expression> must be from 1 to 4 (axis number).

EXPLANATION:
After positioning for the arm of the axis which is specified in <expression 1> (to within effective position tolerance), this command sets the tolerance value of the sub group axis specified in <expression 1> to the value in <expression 2>.

POINT

- *This command is valid only when the sub group has been set in system generation.*
- *If an axis is set when "no axis" has been specified in system generation mode, there will be an error message "Specification mismatch" to remind the user of the conflict in usage. The execution of the program will also be halted.*

EXAMPLE:

```
        'CYCLE WITH DECREASING TOLERANCE
        FOR A=100 TO 20 STEP -20
        GOSUB *CHANGE_TOLE
        MOVE2 P, P0
        MOVE2 P, P1
        NEXT A
        HALT
    *CHANGE_TOLE:
        FOR B=1 TO 4
        TOLE2(B)=A
        NEXT B
        RETURN
```

W A I T Statements

FORMAT 1:

W A I T < D I / D O condition> [, <expression>]

EXPLANATION:
This command will cause the robot to wait for the <DI/DO condition> to be met. The time out time (unit: ms) is set with the second <expression>. When the time out time is set, the robot will still wait for the DI/DO condition to be met before moving, even after the time out time has passed.

FORMAT 2:

WAIT	*ARM [(<expression1>)]* *ARM2 [(<expression2>)]*	

The value of <expression1> must be from 1 to 6 (axis number).
The value of <expression2> must be from 1 to 4 (axis number).

EXPLANATION:
This command waits until completion of robot arm movement (within effective position tolerance).
When an <expression> is not specified, all axes on the main robot or the sub robot are subject to this command. When an <expression> is specified, only the specified axes are subject to this command.

EXAMPLES:

W A I T A = 1 0
---------- Robot will wait until variable A is 10.

W A I T D I 2 ( ) = & B 0 1 0 1 0 1 1 0
---------- Robot will wait until DI(21), (22), (24), (26) are ON, and DI(20), (23), (25), (27) are OFF.

W A I T D I 2 (4 , 3 , 2) = & B 1 0 1
---------- Robot will wait until DI(22) and DI(24) are ON, and DI(23) is OFF.

W A I T D I 2 ( ) = 1 5
---------- Robot will wait until DI(20), (21), (22), (23) are ON and DI(24), (25), (26), (27) are OFF.

W A I T D I (4 7 , 3 5 , 2 7 , 2 0) = & H 0 C
---------- Robot will wait until DI(47), (35) are ON, and DI(27), (20) are OFF.

W A I T D I (3 1) = 1 O R D O (2 1) = 1

---------- Robot will wait until DI(31) is ON, and DO(21) is OFF.

W A I T D I (2 0) = 1 , 1 0 0 0

---------- Robot will wait until DI(20) is ON, and if it does not turn ON after 1 second, the command will end.

W A I T A R M

---------- Robot (main robot) will wait until completion of robot arm movement.

W A I T A R M 2 (2)

---------- Robot (sub robot) will wait until completion of Y-axis movement.

RELATED COMMAND: DRIVE, DRIVE2, DRIVEI, DRIVEI2, MOVE, MIVE2, MOVEI, MOVEI2

W E I G H T Statements (Weight Parameter Setting State ment for Main Robot)

FORMAT:

```
W E I G H T   <expression>
```

EXPLANATION:
This command sets the tip weight parameter of the arm on the main robot to the value of <expression>.

POINT

The robot tip weight is changed. The auxiliary axis is not influenced at all.

EXAMPLE:

```
A = 5
B = 2
C = W E I G H T   ---------- Evacuation
W E I G H T  A
M O V E  P , P 0
W E I G H T  B
M O V E  P , P 1
W E I G H T  C    ---------- Restoration
H A L T
```

W E I G H T 2 Statements (Weight Parameter Setting Statement for Sub Robot)

FORMAT:

```
W E I G H T 2 <expression>
```

EXPLANATION:
This command sets the tip weight parameter of the arm on the sub robot to the value of <expression>.

EXAMPLE:

```
A = 5
B = 2
C = W E I G H T 2---------- Evacuation
W E I G H T 2  A
M O V E 2  P , P 0
W E I G H T 2  B
M O V E 2  P , P 1
W E I G H T 2  C ---------- Restoration
H A L T
```

POINT

This command is valid only when the sub robot has been set in system generation.

W H I L E and W E N D Statements

The WHILE statement will cause an operation to be repeated while the condition specified by <expression> is met.

FORMAT:

```
W H I L E   <expression>
    :
W E N D
```

EXPLANATION:
The WHILE statement will cause the commands between the WHILE and WEND statements to be repeated in order while the condition specified by <expression> is met. If the condition specified by <expression> is not met, the WHILE block is jumped and the program continues from the statement that follows the WEND statement.
It is possible to nest WHILE and WEND statements. When doing this, each WHILE block is ended with the nearest WEND statement.
Statements such as GOTO can be used to jump out of the loop WHILE — WEND.

POINT

If a WEND statements precedes a WHILE statement, there will be an error message "WEND without WHILE" displayed.

EXAMPLES 1:

```
A = 0
WHILE  DI3(0) = 0
    A = A + 1
    MOVE  P,  P0
    MOVE  P,  P1
    PRINT  "COUNTER = "; A
WEND
HALT
```

EXAMPLES 2:

```
A = 0
WHILE  - 1 ----------Since this condition is always true (-1), this is an
                     endless loop.
    A = A + 1
    MOVE  P,  P0
    IF  DI3(0) = 1  THEN  *END
    MOVE  P,  P1
    PRINT  "COUNTER = "; A
    IF  DI3(0) = 1  THEN  *END
WEND
*END
HALT
```

Label Statements

FORMAT:

```
<label>:
```

EXPLANATION:
*This command defines labels that are located at the head of program lines. Labels must begin with a " * ".*

EXAMPLES:

```
* S U B 2 :
* P R O G _ E N D :
```

12 Functions

12-1 Arithmetical Functions

These functions are used in arithmetical expressions.

■ A B S

FORMAT:

```
A B S (<expression>)
```

EXPLANATION:
This function gives the absolute value of the <expression>.

EXAMPLE:

```
A = A B S ( - 3 2 6 . 5 4 )
        ---------- The absolute value of -326.54 (=326.54) is assigned
                   to variable A.
```

■ A C C E L

FORMAT:

```
A C C E L (<expression>)
```

The value of <expression> must be from 1 to 6 (representing an axis).

EXPLANATION:
This function gives the acceleration parameter for the axis (in the main group) specified in the <expression>.

POINT

In "GENERATE" mode, if an axis that has been specified as "no axis" is used in the <expression>, an error message "Specification mismatch" will be displayed and the program will stop.

EXAMPLE:

```
A = A C C E L ( 5 )
        ---------- The acceleration parameter for axis "5" (A) will be
                   assigned to A.
```

■ *ACCEL2*

FORMAT:

```
A C C E L 2 (<expression>)
```

The value of <expression> must be from 1 to 4 (representing an axis).

EXPLANATION:
This function gives the acceleration parameter for the axis (in the sub group) specified in the <expression>.

POINT

- *This function is valid only when the sub group has been set in system generation.*
- *In system generation, if an axis that has been specified as "no axis" is used in the <expression>, an error message "Specification mismatch" will be displayed and the program will stop.*

EXAMPLE:

```
A = A C C E L 2 ( 3 )
        ---------- The acceleration parameter for axis "3" (Z) in the
                   sub group will be assigned to A.
```

■ *ARCH*

FORMAT:

```
A R C H (<expression>)
```

The value of <expression> must be from 1 to 6 (representing an axis).

EXPLANATION:
This function gives the arch position parameter for the axis (in the main group) specified in the <expression>.

POINT

In "GENERATE" mode, if an axis that has been specified as "no axis" is used in the <expression>, an error message "Specification mismatch" will be displayed and the program will stop.

EXAMPLE:

A = A R C H (1)

---------- The arch position parameter for axis "1" (X) will be assigned to A.

■ *A R C H 2*

FORMAT:

A R C H 2 (<expression>)

The value of <expression> must be from 1 to 4 (representing an axis).

EXPLANATION:

This function gives the arch position parameter for the axis (in the sub group) specified in the <expression>.

POINT

- *This function is valid only when the sub group has been set in system generation.*
- *In system generation, if an axis that has been specified as "no axis" is used in the <expression>, an error message "Specification mismatch" will be displayed and the program will stop.*

EXAMPLE:

A = A R C H 2 (1)

---------- The arch position parameter for axis "1" (X) in the sub group will be assigned to A.

■ *A R M T Y P E*

FORMAT:

A R M T Y P E

EXPLANATION:

This function obtains the current hand of the SCARA robot in the main group. The value "0" is obtained for the right hand, and the value "1" for the left hand.

This function is valid only when a SCARA robot is used.

EXAMPLE:

A = A R M T Y P E

---------- The main robot arm type is assigned to variable A.

```
IF  A=0  THEN
    MOVE  P,  P100,  Z=0
        ---------- If variable A is 0 (arm type is right-handed), the
                   main robot moves to point 100 in arch motion.
ELSE
    MOVE  P,  P200,  Z=0
        ---------- If variable A is 1 (arm type is left-handed), the main
                   robot moves to point 200 in arch motion.
ENDIF
```

■ ARMTYPE2

FORMAT:

```
ARMTYPE2
```

EXPLANATION:
This function obtains the current hand of the SCARA robot in the sub group. The value "0" is obtained for the right hand, and the value "1" for the left hand.
This function is valid only when a SCARA robot is used.

EXAMPLE:

```
A=ARMTYPE2
        ---------- The main robot arm type is assigned to variable A.
IF  A=0  THEN
    MOVE  P,  P100,  Z=0
        ---------- If variable A is 0 (arm type is right-handed), the sub
                   robot moves to point 100 in arch motion.
ELSE
    MOVE  P,  P200,  Z=0
        ---------- If variable A is 1 (arm type is left-handed), the sub
                   robot moves to point 200 in arch motion.
ENDIF
```

■ ATN

FORMAT:

```
ATN (<expression>)
```

EXPLANATION:
This function will give the atangent of the value of the <expression>. The resulting values generated may be from -π /2rad to +π /2rad.

EXAMPLES:

*A (0) = A * A T N (Y / X)*

---------- The atangent of Y/X times the variable A will be assigned to place 0 of the array A.

B = 0 . 5

A (0) = A T N (B)

---------- The atangent of the variable B (=0.5) will be assigned to place 0 of the array A.

RELATED FUNCTION: COS, DEGRAD, RADDEG, SIN, TAN

■ A X W G H T

FORMAT:

A X W G H T (<expression>)

The value of <expression> must be from 1 to 6 (representing an axis).

EXPLANATION:

This function gives the axis tip weight parameter for the axis (in the main group) specified in the <expression>.

POINT

- *This command is valid only when the main robot is a MULTI type robot or executing to the main auxiliary axis.*
  Robot type and the auxiliary axes are set at the time of shipment.
- *In system generation, if an axis that has been specified as "no axis" is used in the <expression>, an error message "Specification mismatch" will be displayed and the program will stop.*

EXAMPLE:

A = A X W G H T (1)

---------- The axis tip weight parameter for axis "1" (X) will be assigned to A.

■ A X W G H T 2

FORMAT:

A X W G H T 2 (<expression>)

The value of <expression> must be from 1 to 4 (representing an axis).

EXPLANATION:
This function gives the axis tip weight parameter for the axis (in the sub group) specified in the <expression>.

POINT

- *This function is valid only when the sub group has been set in system generation.*
- *This command is valid only when the sub robot is a MULTI type robot or executing to the sub auxiliary axis.*
  Robot type and the auxiliary axis are set at the time of shipment.
- *In system generation, if an axis that has been specified as "no axis" is used in the <expression>, an error message "Specification mismatch" will be displayed and the program will stop.*

EXAMPLE:

```
A = A X W G H T 2 ( 1 )
        ---------- The axis tip weight parameter for axis "1" (X) in
                   the sub group will be assigned to A.
```

■ C O S

FORMAT:

```
C O S (<expression>)
```

EXPLANATION:
This function will give the cosine of the value of the <expression>. The expression is in units of radians.

EXAMPLES:

```
A ( 0 ) = B * C O S ( C )
        ---------- The cosine of the variable C times the variable B
                   will be assigned to place 0 of the array A.


A ( 1 ) = C O S ( DEGRAD(20) )
        ---------- The cosine of 20.0° will be assigned to place 1 of
                   the array A.
```

RELATED FUNCTION: ATN, DEGRAD, RADDEG, SIN, TAN

■ DEGRAD

FORMAT:

```
DEGRAD (<expression>)
```

EXPLANATION:
This function changes the <expression> into radians. Units are in degrees.

EXAMPLE:

```
A = COS ( DEGRAD ( 30 ) )
---------- The cosine of 30° will be assigned to the array A.
```

RELATED FUNCTION: ATN, COS, RADDEG, SIN, TAN

■ DIST

FORMAT:

```
DIST (<point expression 1>,<point expression 2>)
```

EXPLANATION:
This function gives the distance between the 2 points (X,Y or Z) displayed with <point expression 1> and <point expression 2>. An error is issued when 2 points of a <point expression> are not in the cartesian coordinate system.

EXAMPLE:

```
A = DIST ( P0 , P1 )
---------- The distance between P0 and P1 will assigned to
           the variable A.
```

■ ERR, ERL

FORMAT:

```
ERR
ERL
```

EXPLANATION:
ERR gives the error code of an error that has occurred, and ERL gives the line in which the error occurred.
ERR and ERL are used in error processing routines that are jumped to with the ON ERROR GOTO statement.

EXAMPLES:

```
IF ERR<>&H604 THEN HALT
IF ERL=20 THEN RESUME NEXT
```

RELATED COMMAND: ON ERROR GOTO, RESUME

■ *INT*

FORMAT:

```
I N T (<expression>)
```

EXPLANATION:
This function gives the value of the <expression> after eliminating all decimals, resulting in integers only, and the greatest value that does not exceed the value of the <expression>.

EXAMPLES:

```
A = I N T ( A ( 0 ) )
B = I N T ( -1 . 2 3 3 )----------  -2 is assigned to B.
```

■ *LEN*

FORMAT:

```
L E N (<character string expression>)
```

EXPLANATION:
This function gives the length (number of bytes) of a <character string expression>.

EXAMPLE:

```
B = L E N ( A $ )
```

■ *LSHIFT*

FORMAT:

```
L S H I F T (<expression 1>, <expression 2>)
```

EXPLANATION:
This function shifts the value of <expression 1> to the left, by the number of bits in <expression 2>. 0 is put in the place that is right open by shifting.

EXAMPLE:

```
A = L S H I F T ( & B 1 0 1 1 1 0 1 1 , 2 )
        ---------- &B1011101100 is assigned to A.
```

RELATED FUNCTION: RSHIFT

■ *M C H R E F*

FORMAT:

```
M C H R E F (<expression>)
```

The value of <expression> should be 1 to 6. (axis number)

EXPLANATION:
This function gives the return to origin machine reference for the main group axis which is specified by <expression>.

EXAMPLE:

```
A = M C H R E F  ( 1 )
        ---------- Substitute the return to origin machine reference
                   of the 1st axis of main group (X-axis) for variable
                   A.
```

■ *M C H R E F 2*

FORMAT:

```
M C H R E F 2 (<expression>)
```

The value of <expression> should be 1 to 4. (axis number)

EXPLANATION:
This function gives the return to origin machine reference for the sub group axis which is specified by <expression>.

EXAMPLE:

```
A = M C H R E F 2  ( 3 )
        Substitute the return to origin machine reference of the 3rd axis
    of sub group (Z-axis) for variable A.
```

■ O R D

FORMAT:

```
O R D (<character string expression>)
```

EXPLANATION:
This function gives the character code of the first character of the <string expression>.

EXAMPLE:

```
A = O R D ( " B " )
        ---------- 66(=&H42) is assigned to A.
```

RELATED FUNCTION: CHR$

■ O R G O R D

FORMAT:

```
O R G O R D
```

EXPLANATION:
This function gives the axis sequence parameter which performs return to origin movement for the main group.

EXAMPLE:

```
A = O R G O R D
        ---------- Substitute the return to origin sequence parameter
                   of the main group for variable A.
```

■ O R G O R D 2

FORMAT:

```
O R G O R D 2
```

EXPLANATION:
This function gives the axis sequence parameter which performs return to origin movement for the sub group.

EXAMPLE:

```
A = O R G O R D 2
        ---------- Substitute the return to origin sequence parameter
                   of the sub group for variable A.
```

■ O U T P O S

FORMAT:

```
O U T P O S (<expression>)
```

The value of the <expression> must be 1 to 6 (representing an axis).

EXPLANATION:
This function gives the axis out effective position parameter for the axis (in the main group) specified in the <expression>.

POINT

In "GENERATE" mode, if an axis that has been specified as "no axis" is used in the <expression>, an error message "Specification mismatch" will be displayed and the program will stop.

EXAMPLE:

```
A = O U T P O S ( 3 )
        ---------- The out possible parameter for axis "3" (Z) will be
                   assigned to A.
```

■ O U T P O S 2

FORMAT:

```
O U T P O S 2 (<expression>)
```

The value of the <expression> must be 1 to 4 (representing an axis).

EXPLANATION:
This function gives the axis out effective position parameter for the axis (in the sub group) specified in the <expression>.

POINT

- *This function is valid only when the sub group has been set in system generation.*
- *In system generation, if an axis that has been specified as "no axis" is used in the <expression>, an error message "Specification mismatch" will be displayed and the program will stop.*

EXAMPLE:

```
A = O U T P O S 2 ( 3 )
        ---------- The out possible parameter for axis "3" (Z) in the
                   sub group will be assigned to A.
```

■ *R A D D E G*

FORMAT:

```
R A D D E G (<expression>)
```

EXPLANATION:
This function changes the <expression> into degrees. Units are in radians.

EXAMPLE:

```
L O C R ( P 0 ) = R A D D E G ( A T N ( B ) )
        ---------- The atangent of the variable B will changed to de-
                   grees and assigned to R data for the P0.
```

RELATED FUNCTION: ATN, COS, DEGRAD, SIN, TAN

■ *R S H I F T*

FORMAT:

```
R S H I F T (<expression 1>, <expression 2>)
```

EXPLANATION:
This function shifts the value of <expression 1> to the right, by the number of bits in <expression 2>. 0 is put in the place that is left open by shifting.

EXAMPLE:

```
A = R S H I F T ( & B 1 0 1 1 1 0 1 1 , 2 )
        ---------- &B00101110 is assigned to A.
```

RELATED FUNCTION: LSHIFT

■ *S I N*

FORMAT:

```
S I N (<expression>)
```

EXPLANATION:
This function will give the sine of the value of the <expression>. The expression is in units of radians.

EXAMPLES:

*A (0) = S I N (B * 2 + C)*

*---------- The cosine of the value of B*2+C will be assigned to place 0 of array A.*

PAI = 3.141592

A (1) = S I N (DEGRAD(30))

---------- The cosine of 30° will be assigned to place 1 of array A.

RELATED FUNCTION: ATN, COS, DEGRAD, RADDEG, TAN

■ S Q R

FORMAT:

```
S Q R (<expression>)
```

EXPLANATION:
This function gives the square root of the value of the expression.

POINT

A negative value in the <expression> will cause an error.

EXAMPLE:

A = S Q R (X ^ 2 + Y ^ 2)

---------- The square root of X^2+Y^2 will be assigned to A.

■ T A N

FORMAT:

```
T A N (<expression>)
```

EXPLANATION:
This function gives the tangent of the value of the <expression>. The unit of the expression is radians.

EXAMPLES:

A (0) = B - T A N (C)

---------- The tangent of variable C is subtracted from variable B and the result is assigned to place 0 of array A.

A (1) = T A N (DEGRAD(20))

---------- The tangent of 20° is assigned to place 1 of array A.

RELATED FUNCTION: ATN, COS, DEGRAD, RADDEG, SIN

■ *TIMER*

FORMAT:

```
TIMER
```

EXPLANATION:
This function gives the time according to the robot's timer, in seconds, counting from 12:00 midnight as "0:00". It is used to measure operation time during program execution, as well as for other uses.

POINT

The timer is set in "SYSTEM" mode during initializing procedures (refer to the separate "12-4 Initializing" in Chapter 4 on the User's Manual).

EXAMPLE:

```
A% = TIMER
FOR  B = 1  TO  10
MOVE  P,  P0
MOVE  P,  P1
NEXT
A% = TIMER - A%
PRINT  A% / 60 ; " : " ; A%  MOD  60
HALT
```

RELATED FUNCTION: TIME$

■ *TOLE*

FORMAT:

```
TOLE (<expression>)
```

The value of <expression> must be from 1 to 6 (representing an axis).

EXPLANATION:
This function gives the tolerance parameter for the main group axis specified in the <expression>.

POINT

In "GENERATE" mode, if an axis that has been specified as "no axis" is used in the <expression>, an error message "Specification mismatch" will be displayed and the program will stop.

EXAMPLE:

A = T O L E (2)

---------- The tolerance parameter for axis "2" (Y) in the main group will be assigned to A.

■ *T O L E 2*

FORMAT:

T O L E 2 (<expression>)

The value of <expression> must be from 1 to 4 (representing an axis).

EXPLANATION:

This function gives the tolerance parameter for the sub group axis specified in the <expression>.

POINT

- *This function is valid only when the sub group has been set in system generation.*
- *In system generation, if an axis that has been specified as "no axis" is used in the <expression>, an error message "Specification mismatch" will be displayed and the program will stop.*

EXAMPLE:

A = T O L E 2 (2)

---------- The tolerance parameter for axis "2" (Y) in the sub group will be assigned to A.

■ *V A L*

FORMAT:

V A L (<character string expression>)

EXPLANATION:

This function gives the actual value of the character string of numerical characters specified by the <character string expression>.

The value may be expressed in integer format (binary, decimal, hexadecimal), or real number format (regular decimal point format, exponent format).

If the first character of the character string is "+", "-", "&" or anything other than a numeric character, the VAL function value becomes 0. If there are non-numerical characters or spaces elsewhere in the character string, all succeeding characters are ignored by this function. However, in the case of hexadecimal numbers, A to F are considered part of the numerical value.

EXAMPLE:

A = V A L (" & B 1 0 0 0 0 1 ")

RELATED FUNCTION: STR$

■ *W E I G H T*

FORMAT:

W E I G H T

EXPLANATION:

This function gives the weight parameter for the main robot.

EXAMPLE:

A = W E I G H T ---------- The weight parameter is assigned to A.

■ *W E I G H T 2*

FORMAT:

W E I G H T 2

EXPLANATION:

This function gives the weight parameter for the sub robot.

EXAMPLE:

A = W E I G H T 2 ---------- The weight parameter is assigned to A.

POINT

This function is valid only when the sub group has been set in system generation.

12-2 Character String Functions

Character string functions are used as elements in character string expressions.

■ C H R $

FORMAT:

```
C H R $ (<expression>)
```

EXPLANATION:
This function gives the character for the character code which results from <expression>. An error of "Illegal function call" occurs when the value of the <expression> is not between 0 to 255.

EXAMPLE:

```
A $ = C H R $ ( 6 5 )
        ---------- The character "A" is assigned to character string
                   A$.
```

RELATED FUNCTION: ORD

■ D A T E $

FORMAT:

```
D A T E $
```

EXPLANATION:
This function gives the data in the following format: yy/mm/dd (last two digits of the year, month, day).

POINT

The data is set in "SYSTEM" mode during initializing procedures (refer to the separate "12-4 Initializing" in Chapter 4 on the User's Manual).

EXAMPLES:

```
A $ = D A T E $
P R I N T  D A T E $
```

RELATED FUNCTION: TIME$

■ *LEFT$*

FORMAT:

```
L E F T $ (<character string expression>, <expression>)
```

EXPLANATION:
This function gives the string composed of a specific number of characters from the left side of the <character string expression>. The number of characters in the string is specified by the value of <expression>. The value of <expression> must be between 0 and 75, otherwise an error will result. If the value of <expression> is 0, then LEFT$ will be a null string. If the value of <expression> is equal to or greater than the length of <character string expression>, the LEFT$ string will be the same as the <character string expression>.

EXAMPLE:

```
B $ = L E F T $ ( A $ ,  4 )
```

RELATED FUNCTION: MID$, RIGHT$

■ *MID$*

FORMAT:

```
M I D $ (<character string expression>, <expression 1>[, <expression 2>])
```

EXPLANATION:
This function gives the string composed of a specific number of characters from the <character string expression>. The number of characters in the string is specified by the value of <expression 2>. <expression 1> is the character from which the MDI$ string will begin. The value of <expression 2> or <expression 1> must be between 0 and 75, otherwise an error will result. If <expression 2> is eliminated, or if the number of characters to the right of the character of <expression 1> is less than the value of <expression 2>, the MID$ string will consist of all characters to the right of the character expressed by <expression 1>. If <expression 1> is longer than the character string, MID$ will be a null string.

EXAMPLE:

```
B $ = M I D $ ( A $ ,  2 ,  4 )
```

RELATED FUNCTION: LEFT$, RIGHT$

■ RIGHT$

FORMAT:

```
RIGHT$ (<character string expression>, <expression>)
```

EXPLANATION:
This function gives the string composed of a specific number of characters from the right side of the <character string expression>. The number of characters in the string is specified by the value of <expression>. The value of <expression> must be between 0 and 75, otherwise an error will result. If the value of <expression> is 0, then RIGHT$ will be a null string. If the value of <expression> is equal to or greater than the length of <character string expression>, the RIGHT$ string will be the same as the <character string expression>.

EXAMPLE:

```
B$ = RIGHT$ ( A$ ,  4 )
```

RELATED FUNCTION: LEFT$, MID$

■ STR$

FORMAT:

```
STR$ (<expression>)
```

EXPLANATION:
This function changes the value of <expression> into a character string. The value of <expression> may be either an integer or real number.

EXAMPLE:

```
B$ = STR$ ( 10.01 )
```

RELATED FUNCTION: VAL

■ TIME$

FORMAT:

```
TIME$
```

EXPLANATION:
This function gives the time in the following format: hh:mm:ss (hours, minutes, seconds).

POINT

The clock is set in "SYSTEM" mode during initializing operations (refer to the separate "12-4 Initializing" in Chapter 4 on the User's Manual).

EXAMPLES:

```
A$ = TIME$
PRINT  TIME$
```

RELATED FUNCTION: DATE$, TIMER

12-3 Point Functions

Point functions are used as elements of point expressions only.

■ J T O X Y

FORMAT:

```
J T O X Y (<point expression>)
```

EXPLANATION:
This function converts the <point expression> from joint coordinate data (unit: pulses) to Cartesian coordinate data (unit: mm, Åã) for the main group.

POINT

When executed, the data is converted based on the standard coordinate system, shift coordinate system and hand definition in effect.

EXAMPLE:

```
P 1 0 = J T O X Y ( W H E R E )
        ---------- Current position is converted to Cartesian coordi-
                    nate data.
```

RELATED FUNCTION: XYTOJ

■ J T O X Y 2

FORMAT:

```
J T O X Y 2 (<point expression>)
```

EXPLANATION:
This function converts the <point expression> from joint coordinate data (unit: pulses) to Cartesian coordinate data (unit: mm, Åã) for the sub group.

POINT

- *This function is valid only when the sub group has been set in system generation.*
- *When executed, the data is converted based on the standard coordinate system, shift coordinate system and hand definition in effect.*

EXAMPLE:

P 1 0 = J T O X Y 2 (W H E R E 2)

---------- Current position is converted to Cartesian coordinate data.

RELATED FUNCTION: XYTOJ2

■ *W H E R E*

FORMAT:

```
W H E R E
```

EXPLANATION:

This function gives the current position of the arm (in the main group) in joint coordinates.

EXAMPLE:

P 1 0 = W H E R E

---------- Current position is assigned to point 10. Point 10 is defined in joint coordinates (pulses).

■ *W H E R E 2*

FORMAT:

```
W H E R E 2
```

EXPLANATION:

This function gives the current position of the arm (in the sub group) in joint coordinates.

EXAMPLE:

P 1 0 = W H E R E 2

---------- Current position is assigned to point 10. Point 10 is defined in joint coordinates (pulses).

POINT

This function is valid only when the sub group has been set in system generation.

■ *X Y T O J*

FORMAT:

```
X Y T O J (<point expression>)
```

EXPLANATION:
This function converts the <point expression> from Cartesian coordinate data (unit: mm, °) for the main group to joint coordinate data (unit: pulses).

POINT

When executed, the data is converted based on the standard coordinate system, shift coordinate system and hand definition currently in effect. Please note that when using a SCARA type robot, the converted result differs depending on whether right-hand or left-hand was specified.

EXAMPLE:

```
P 1 0 = X Y T O J ( P 1 0 )
        ----------------- Point 10 is converted to joint coordinate data.
```

RELATED FUNCTION: JTOXY

■ *X Y T O J 2*

FORMAT:

```
X Y T O J 2 (<point expression>)
```

EXPLANATION:
This function converts the <point expression> from Cartesian coordinate data (unit: mm, °) for the sub group to joint coordinate data (unit: pulses).

POINT

- *This function is valid only when the sub group has been set in system generation.*
- *When executed, the data is converted based on the standard coordinate system, shift coordinate system and hand definition currently in effect. Please note that when using a SCARA type robot, the converted result differs depending on whether right-hand or left-hand was specified.*

EXAMPLE:

P 1 0 = X Y T O J 2 (P 1 0)

---------- Point 10 is converted to joint coordinate data.

RELATED FUNCTION: JTOXY2

13 Multi-tasking

Multi-tasking is not essential for normal applications. It is used to program for more complex tasks.
To use the multi-tasking capability, be sure to read this section thoroughly.

13-1 Outline

Multi-tasking can be used to perform complex tasks simultaneously.
The program is divided up into tasks, for example, robot movement and numerical calculation can be performed at the same time, shortening cycle time.
A maximum of 8 tasks may be performed simultaneously. Task numbers are given from 1 to 8. Task priority is assigned in order from 1 to 8. Task 1 is the main task, and is automatically put on WAIT status when the power of the robot is turned on.

13-2 Task Status

There are 5 types of status for tasks:

(1) STOP
The program is not being executed but is in existence. If the program is reset, tasks 2 to 8 are put into this state.

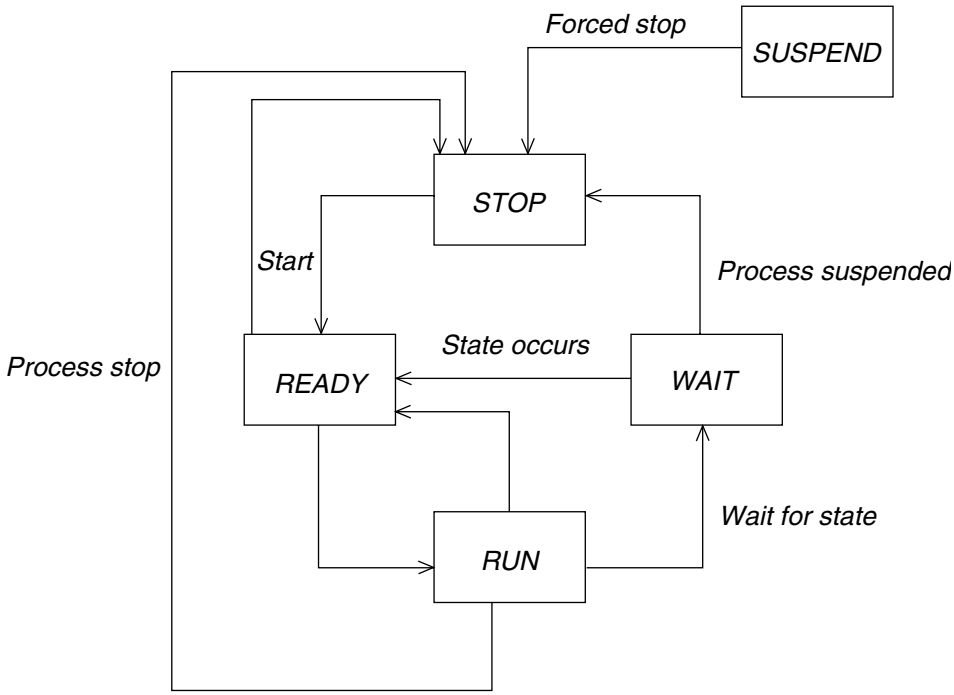
(2) RUN
In this state, the program is executed.

(3) READY
It is possible to the program to be run in this state, but another task is being executed.
When a number of tasks are in the READY state, the task with the highest priority is but into RUN, and the other tasks are kept in READY.

(4) WAIT
In this state, the program is waiting for a certain state. As long as this state does not occur, the task will not be executed. When the program is reset, the robot returns to task 1 (and waits for the START *key).*

(5) SUSPEND
This state is brought about by the SUSPEND command. The robot is forced to suspend execution of the task. Nothing will be done until the RESTART command is given.

Task Status Flow



13-3 Task Definition

Tasks are defined by placing a label at the head of the program. The label format is the same as for conventional labels.

EXAMPLE:

```
*IOTASK: ---------------------- Task name is "IOTASK"
    WAIT  A=1
    MOVE  P, P100
    DO2(0)=1
    DO2(1)=1
    WAIT  DI4(0)=1
    DO2(1)=0
    DO2(0)=0
    WAIT  DI4(1)=1
    EXIT  TASK
```

13-4 Starting Tasks

The START command is used to start tasks. All tasks except the main task will start.
Status flow is as follows:

Priority:
When: started side> start side
Started side → RUN
Start side → READY
When: started side < start side
Started side ← READY
Start side ← RUN

13-5 Task Status Flow

When the task is put into the WAIT state with the WAIT command, the following happens:
The task for which WAIT command is executed → WAIT
The highest priority task in ready state → RUN
When a task is being run and another task's condition is met
Other task in wait state → READY
And when a task being run whose priority is lower than that of another task in READY state.
Task during RUN → READY
Task in READY state → RUN

CAUTION

- ***When a task is in the RUN state, the lower task will not be run unless otherwise a command which effects the WAIT state is included.***
- ***When a task is in the RUN state, the higher task checks transition to the RUN state at certain time intervals even if no command which effects the WAIT state is included. When a higher task sets to the RUN state, the current task moves to the READY state.***
- ***When a task is in the WAIT state, events regarding this task are checked at certain time intervals. The following events may take place.***
  1. ***Under the condition that multiple tasks are in the WAIT state, if the event of a lower task is first checked before checking a higher task, the lower task sets to the RUN state.***
  2. ***Under the condition that multiple tasks are in the WAIT state at the same event, the lower priority task first sets to the RUN state.***

Commands which effect the WAIT state are as follows:
Event
1) MOVE, MOVEI, DRIVE, DRIVEI,
MOVE2, MOVEI2, DRIVE2, DRIVEI2
---------- Wait for positioning to be complete
2) DELAY ---- Wait for time
3) WAITWait for condition

4) SEND ------ Wait for reception
5) TOLE, CHANGE, SHIFT, RIGHTY, LEFTY,
TOLE2, CHANGE2, SHIFT2, RIGHTY2, LEFTY2
---------- Wait for positioning to be complete
6) PRINT ----- Wait for print buffer to be empty
7) INPUT----- Wait for key input

If in the case of the WAIT command of 3), the condition is met from the start, or if in 5) the positioning is already complete, the task is never put in the wait state.

13-6 Task Completion

It is possible to end a task during RUN by using the EXIT TASK command. The status flow is as follows:

The task for which EXIT TASK is executed → STOP
Other Tasks
The highest priority task in READY state → RUN

This command cannot be used in the main task.

13-7 Completion of Other Tasks

The CUT command is used to end other tasks. The status flow is as follows:

Task that is cut → RUN
Defined task → STOP

This command is used to end tasks other than the main task.

13-8 Tasks Suspension

The SUSPEND command suspends other tasks. When this command is executed, the target task alone is suspended. Other tasks are executed as they are. The status flow is as follows:

Task that is suspended → RUN
Defined task → SUSPEND

This command cannot be used in the main task.

13-9 Starting Tasks

To start other tasks that are suspended, the RESTART command is used. When this command is executed, the suspended task is executed.
The status flow is as follows:

Restarted task → RUN
Defined task → in state at time of suspension

This command cannot be used in the main task.

13-10 Stopping Programs

Programs stop in any one of the following cases:

(1) HALT command is executed
Except for the main task, after all tasks are ended (STOP state), the program is reset. The main task is put in the WAIT state (waiting for the START *key).*

(2) HOLD command is executed, or the STOP *key is pressed, or an INTERLOCK signal is input*
When the robot is moving, the arm is slowed down and stopped, and all the tasks are put into the WAIT state. When the robot is started again, all suspended tasks are restarted.

(3) When the emergency stop switch is pressed or the emergency stop signal is input
When the robot is moving, the arm is stopped suddenly, and all the tasks are put into the WAIT state. When the robot is started again, all suspended tasks are restarted.

CAUTION

In the case of (2) or (3), after the program has been stopped, the controller power is turned on again. If the program is not reset, all programs will be run from the state they were in before the power was cut. However, the output and internal output ports are reset.

13-11 Program List Changing

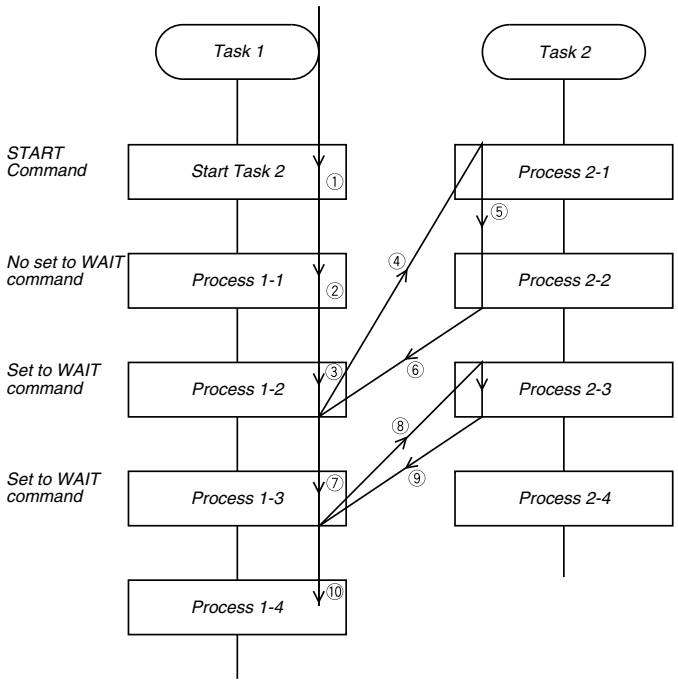
After stopping a program that has many tasks, it is possible to display the status of each task. For details, refer to the separate "9-4 Switching Task Displays" in Chapter 4 on the User's Manual. After changing the program list, execute STEP operation, NEXT, or SKIP operation and the specified task will be run.

POINT

During continuous operation all the tasks are executed. However, with STEP operation, NEXT, or SKIP operation, only the task displayed is executed.

13-12 Program Execution Sequence

Program execution sequence when multi-tasking is as shown below:



① *Task 2 sets to READY with START command.*
② *Execute process 1-1 (not set to WAIT command), and then sets to READY.*
③ *Execute process 1-2 (set to WAIT command), and then sets to WAIT.*
④ *Task 2 sets to RUN, process 2-1 is executed and then sets to READY.*
⑤ *Process 2-2 is executed with task 2 in RUN and then sets to READY since the task 1 WAIT continues.*
⑥ *Returns to process 1-2 since task 1 is in READY.*
⑦ *Task 1 is in RUN, process 1-3 is executed (set to WAIT command) and then sets to WAIT.*
⑧ *Task 2 is in RUN, process 2-3 is executed and then sets to READY.*
⑨ *Returns to process 1-3 since task 1 is in READY.*
⑩ *Process 1-4 is executed with task 1 in READY, sets to RUN since the task 1 is in READY continues.*

13-13 Common Use of Variables

All global variables, point variables and shift variables can be used in tasks. Therefore caution must be used when the same variables are used in several tasks.
For instance, in the example below do not use the same variables of task 1 and task 2 in the <control variable> of the FOR and NEXT statements.

EXAMPLE:

```
   START *TASK2,T2
*TASK1:
      FOR A=1 TO 10
      MOVE P,P[A]
      NEXT A
      GOTO *TASK1
*TASK2:
      FOR A=1 TO 10
      WAIT DI3()=A
      PRINT "A=";A
      NEXT A
      GOTO *TASK2
```

Variable A is written by both task 1 and task 2.
In this case change the variable name or use a local variable. The example below illustrates how a local variable is used.
*Since variable A for sub-procedure *MOV and variable A for sub-procedure *INPTEST are local variables, they are handled as separate variables.*

```
   START *TASK2,T2
*TASK1:
      CALL *MOV
      GOTO *TASK1
*TASK2:
      CALL *INPTEST
      GOTO *TASK2
SUB *MOV
  FOR A=1 TO 10
  MOVE P,P[A]
  NEXT A
END SUB
SUB *INPTEST
  FOR A=1 TO 10
  WAIT DI3()=A
  PRINT "A=";A
  NEXT A
END SUB
```

14 Command Statement List

Types of Statements	Format	Example	Memory Usage [bytes]	Executable Condition (*1)
Array Variable Declaration Statement	DIM <array definition> [, <array definition>,.....]	DIM A%(10) DIM B(2, 2, 2), C%(3,2) DIM C$ (5)	11 24 11	6
Function Definition Statements	DEF FN<name>[% | ! | $][(<parameter>, <parameter>...)]=<function definition expression>	DEF FNAPAI=3.141592 DEF FNASIN(X)=ATN(X/SQR(-X^2+1))	13 29	6
Point Definition	Pm=x y z r a b	P123=100.00 200.00 0.00 0.00 0.00 0.00	31	1
Shift Coordinate Definition	Sm=x y z r	S0=-123.45 123.45 123.45 123.45	21	1
Definition	SUB[(<parameter>[, <parameter>...])] : END SUB	SUB *DISTANCE(X!, Y!, D!) : END SUB	19 4	6
Referring to Global Variables	SHARED <simple variable>[( )][, <simple variable>[( )]...]	SHARED X!, Y!, D!( )	12	6
Referring to External Symbols	DECLARE <label>[, <label>...] DECLARE SUB <label>[(<parameter>[, <parameter>....])]	DECLARE ÅñDISTANCE, *AREA DECLARE SUB *COMPARE DECLARE SUB *AREA(X!, Y!)	14 11 17	6
Main Robot Hand Definition	HAND Hn=<1st parameter><2nd parameter><3rd parameter>[R]	HAND H1= 0 150.0 0.0 HAND H2= 45.0 20.0 0.0 R	20 20	1
Sub Robot Hand Definition	HAND2 Hn=<1st parameter> <2nd parameter> <3rd parameter> [R]	HAND2 H5= 0 150.0 0.0 HAND2 H6= 45.0 20.0 0.0 R	21 21	1
Arithmetical Assignation Statement	[LET] <variable>=<expression>	A=10 B(0)=10.05 LOCX(P1)=A(1) LOCX(S1)=100.00	13 20 21 17	1
Character String Assignation Statement	[LET] <character string variable>=<character string expression>	A$="YAMAHA" B$ (1) =A$+" ROBOT"	24 29	1
Point Assignation Statement	[LET] Pm=<point expression>	P1=P100 P[A]=P200+P5 P[START_POINT]=P300/2-4*P3	13 23 34	1 6

Types of Statements	Format	Example	Memory Usage [bytes]	Executable Condition (*1)
Shift Assignation Statement	[LET] Sm=<shift expression>	S1=S0 S[A]=-S1+S2	11 22	1
Jumping	GOTO | GO TO <label>	GOTO * LOOP	8	6
Decisions	IF<expression>THEN <label> | <expression> [ELSE <label> | <expression>]	IF A=1 THEN * PRGEND IF A=1 THEN * L1 ELSE * L2 IF A=1 THEN PRINT "OK" IF A=1 THEN MOVE P, P1 ELSE MOVE P, P2	21 26 25 36	6

IF	<expression> [THEN]
	:
	:
	[ELSE]
	:
	:
	ENDIF
		IF DI3(1)=1 THEN
		MOVE P, P1
		DO(30)=1
		ELSE
		MOVE P, P2
		DO(30)=0
		ENDIF
			19
			
			6
			
			4
				6

Multiple Decisions	SELECT[CASE]<expression> CASE<expression list 1> [Block 1] [CASE<expression list 2> [Block 2]] : : [CASE ELSE [[Block n]] END SELECT	SELECT CASE DI3() CASE 1 CALL *EXEC(1,10) CASE 2 CALL *EXEC(21,30) CASE ELSE END SELECT	10 10 10 6 4	6
Subroutines	GOSUB | GO SUB <label> : : RETURN	DECLARE * INITIALIZE GOSUB * SUBROUTIN GO SUB * INITIALIZE : RETURN	 8 8 4	6
Jumping on Condition	ON <expression> GOTO | GO TO <label> [, <label>...]	ON A GOTO * L10, * L20, * L30	22	6
Conditional Subroutine	ON <expression> GOSUB | GO SUB <label> [, <label>, ...]	ON A GOSUB * SB10, * SB20, * SB30	22	6
Loops	FOR <variable>=<expression 1> TO <expression 2> [STEP <expression 3>] : NEXT <variable>	FOR A=10 TO 4 STEP -2 : NEXT A	24 6	6
		FOR B(0)=0.1 TO 0.5 STEP 0.1 : NEXT B(0)	34 6	6
End Loops	EXIT FOR	EXIT FOR	4	6

Types of Statements	Format	Example	Memory Usage [bytes]	Executable Condition (*1)
Conditional	Loops WHILE <expression> : WEND	WHILE A>10 WHILE DI1( )=&B10111100 : WNED	14 14 6	6
Calling Procedure	CALL <label>[(parameter, [, parameter, ...])]	CALL * DISTANCE CALL * AREA(2.5, X!, REF Y!)	11 41	6
Ending Procedure	EXIT SUB	EXIT SUB	4	6
Error Processing	ON ERROR {GOTO <label> / 0} : RESUME [{NEXT / <label>}]	ON ERROR GOTO * ER1 ON ERROR GOTO 0 : RESUME RESUME NEXT RESUME * L1	9 5 5 6 9	6
Temporary Suspension	HOLD [{<expression> / <character string>}]	HOLD HOLD "ERROR STOP"	5 19	6
Ending Movement	HALT [{<expression> / <character string>}]	HALT HALT "PROGRAM STOP"	5 21	6
Program Switching	SWI <program name>	SWI <ABC>	14	2
Main Robot Absolute Movement	MOVE {P / L / C}, <point definition>[, <, option> [, <option>...]]	MOVE P, P100 MOVE L, 100.00 100.00 200.00 0.00 0.00 0.00 MOVE P, P[A], P10+P20, S=70, Z=0 MOVE P, P1, STOPON DI3(0)=1 MOVE C, P1, P2, S=50 MOVE L, P1, STOPON DI3(0)=1	12 31 43 25 25 25	4
Main Robot Relative Movement	MOVEI P, <point definition>[, <, option>[, <option>...]]	MOVEI P, P100 MOVEI P, 100.00 100.00 200.00 0.00 0.00 0.00 MOVEI P, P[A], P10+P20, S=70	12 31 35	1
Main Group Absolute Axis Unit Movement	DRIVE(<expression>, {<expression> / <point expression>}) [, (<expression>, {<expression>)... / <point expression>}] [, {S / SPEED} =<expression>]	DRIVE(1,100.00) DRIVE(2,100.00),(3,50.00) DRIVE(A, P10), S=10	15 25 20	1
Main Group Relative Axis Unit Movement	DRIVEI(<expression>, {<expression> / <point expression>}) [, (<expression>, {<expression>)... / <point expression>}] [, {S / SPEED} =<expression>]	DRIVEI(1,100.00) DRIVEI(2,100.00),(3,50.00) DRIVEI(A, P10), S=10	15 25 20	1
Main Group SERVO Control	SERVO {ON / OFF / FREE} [(<expression>)]	SERVO ON SERVO OFF SERVO FREE(2)	6 6 10	1
Waiting for Condition	WAIT <condition expression>[, <expression>] WAIT {ARM / ARM2} [(<expression>)]	WAIT DI2( )=&B10101011 WAIT DI2(3, 1)=&B10, 1000 WAIT DI(17, 15, 13, 10)=&H0C WAIT DO2(3, 1)=&B10 AND DI1(7)=1 WAIT ARM WAIT ARM2(1)	14 20 19 26 8 11	6

Types of Statements	Format	Example	Memory Usage [bytes]	Executable Condition (*1)
Sub Robot Absolute Movement	MOVE2 P, <point definition>[, <, option>[, <option>...]]	MOVE2 P, P100 MOVE2 P, 100.00 100.00 200.00 0.00 0.00 0.00 MOVE2 P, P[A], P10+P20, S=70, Z=0 MOVE2 P, P1, STOPON DI3(0)=1	13 32 44 26	1
Sub Robot Relative Movement	MOVEI2 P, <point definition>[, <, option>[, <option>...]]	MOVEI2 P, P100 MOVEI2 P, 100.00 100.00 200.00 0.00 0.00 0.00 MOVEI2 P, P[A], P10+P20, S=70	13 32 36	1
Sub Group Absolute Axis Unit Movement	DRIVE2(<expression>, <expression> | <point expression>) [, (<expression>, <expression>)...] | <point expression>] [, S | SPEED =<expression>]	DRIVE2(1,100.00) DRIVE2(2,100.00),(3,50.00) DRIVE2(A, P10), S=10	16 26 21	1
Sub Group Relative Axis Unit Movement	DRIVEI2(<expression>, <expression> | <point expression>) [, (<expression>, <expression>)...] | <point expression>] [, S | SPEED =<expression>]	DRIVEI2(1,100.00) DRIVEI2(2,100.00),(3,50.00) DRIVEI2(A, P10), S=10	16 26 21	1
Sub Group SERVO Control	SERVO2 ON | OFF | FREE [(<expression>)]	SERVO2 ON SERVO2 OFF SERVO2 FREE(2)	7 7 11	1
DO Output	<DO variable>=<expression>	DO2( )=&B10111000 DO2(7, 5)=2 DO(27, 25, 23, 20)=&H0C	13 15 18	1
MO Output	<MO variable>=<expression>	MO( )=&B10111000 MO(7, 5)=2 MO(27, 25, 23, 20)=&H0C	13 15 18	1
TO Output	<TO variable>=<expression>	TO0( )=&B10111000 TO0(7, 5)=2 TO(07, 05, 03, 00)=&H0C	13 15 18	1
LO Output	<LO variable>=<expression>	LO0( )=&B10111000 LO0(7, 5)=2 LO(07, 05, 03, 00)=&H0C	13 15 18	1
SET Statements	SET <DO variable> | <MO variable> | <TO variable> | <LO variable> [, <expression>]	SET DO2(7, 5, 0) SET MO(27, 25, 23, 20), 1000 SET TO0() SET LO0(7, 5)	10 17 8 10	3
RESET Expressions	RESET <DO variable> | <MO variable> | <TO variable> | <LO variable>	RESET DO2( ) RESET MO(27, 25, 20) RESET TO0(7, 5, 6, 1) RESET LO(07, 05, 00)	8 13 10 13	1
PRINT Statements	PRINT [<expression>][, | ;] <expression>][, | ;]	PRINT "COUNT=";C, "TIME=";T PRINT A$ PRINT "P10=";P10 PRINT S[A]	33 8 17 14	1

Types of Statements	Format	Example	Memory Usage [bytes]	Executable Condition (*1)
INPUT Statements	INPUT [<prompt> ; ,] <variable>[, <variable>...]	INPUT A INPUT "FILE NAME=";F_NAME$ INPUT "P10=";P10 INPUT S0, S1	12 23 17 10	1
Communication	SEND <File 1> TO <File 2>	SEND PGM TO CMU SEND <000000> TO CMU SEND PNT TO SCR SEND PRM TO SCR	11 20 11 11	1
Delay Statements	DELAY <expression>	DELAY 1000 DELAY A * 10	8 16	6
Main Robot Speed Setting Statements	SPEED <expression>	SPEED 70 SPEED A * 10	8 16	1
Main Group Output Effective Position Setting	OUTPOS (<expression>)=<expression>	OUTPOS(1)=100	12	1
Main Group Tolerance Setting	TOLE (<expression>)=<expression>	TOLE(1)=10	12	5
Main Group Acceleration Setting	ACCEL <expression> ACCEL (<expression>)=<expression>	ACCLE 100 ACCEL(1)=100	8 13	1
Main Robot Weight Parameter Setting	WEIGHT <expression>	WEIGHT 10 WEIGHT A	8 12	1
Main Group Axis Tip Weight Setting	AXWGHT (<expression>)=<expression>	AXWGHT(1)=100 AXWGHT(ANO)=B	13 13	1
Main Group Arch Position Setting	ARCH (<expression>)=<expression>	ARCH(1)=100 ARCH(ANO)=B	12 20	1
Main Robot Automatic Speed Setting	ASPEED <expression>	ASPEED 70 ASPEED A * 10	8 16	1
Sub Robot Speed Setting Statements	SPEED2 <expression>	SPEED2 70 SPEED2 A * 10	9 17	1
Sub Group Output Effective Position Setting	OUTPOS2 (<expression>)=<expression>	OUTPOS2(1)=100	12	1
Sub Group Tolerance Setting	TOLE2 (<expression>)=<expression>	TOLE2(1)=10	12	5
Sub Group Acceleration Setting	ACCEL2 <expression> ACCEL2 (<expression>)=<expression>	ACCLE2 100 ACCEL2(1)=100	8 13	1 1

Types of Statements	Format	Example	Memory Usage [bytes]	Executable Condition (*1)
Sub Robot Weight Parameter Setting	WEIGHT2 <expression>	WEIGHT2 10 WEIGHT2 A	8 12	1
Sub Group Axis Tip Weight Setting	AXWGHT2 (<expression>)=<expression>	AXWGHT2(1)=100 AXWGHT2(ANO)=B	13 13	1
Sub Group Arch Position Setting	ARCH2 (<expression>)=<expression>	ARCH2(1)=100 ARCH2(ANO)=B	12 20	1
Sub Robot Automatic Speed Setting	ASPEED2 <expression>	ASPEED 70 ASPEED A * 10	8 16	1
Communica-tion Mode Selection	ONLINE OFFLINE	ONLINE OFFLINE	4 4	1
Main Robot Shift Coordinate Selection	SHIFT <shift variable>	SHIFT S1 SHIFT S[A]	7 10	5
Main Robot Arm Selection	LEFTY RIGHTY	LEFTY RIGHTY	4 4	5
Main Robot Hand Selection	CHANGE Hn	CHANGE H2	6	5
Sub Robot Shift Coordinate Selection	SHIFT2 <shift variable>	SHIFT2 S1 SHIFT2 S[A]	8 11	5
Sub Robot Arm Selection	LEFTY2 RIGHTY2	LEFTY2 RIGHTY2	5 5	5
Sub Robot Hand Selection	CHANGE2 Hn	CHANGE2 H2	7	5
Task Starting	START <label>, Tn	START * IOTASK, T2	9	7
Task Stopping	CUT Tn	CUT T2	6	7
Task Suspension	SUSPEND Tn	SUSPEND T2	6	7
Task Restarting	RESTART Tn	RESTART T2	6	7
Task Ending	EXIT TASK	EXIT TASK	4	7
Comments	REM | ' <comment>	'SUBROUTINE ***	4	6
Label Statements	* <label>:	* L10:	4	6

CAUTION

◯ ***(*1) executable condition***

1 : Commands which can be executed with direct command and online commands.

2 : Commands which can be executed only in task 1 (main task) (also includes condition 1).

3 : Commands with an operand, a portion of which cannot be executed with direct command and online commands (also includes condition 1).

4 : MOVE L and MOVE C commands which can be executed only when an arithmetic processor is provided (also includes condition 1). Moreover, MOVE L and MOVE C commands can be executed only in task 1 (main task). STOPON (option) cannot be executed with direct command and online commands.

5 : Commands which can be executed after positioning (also includes condition 1).

6 : Commands which cannot be executed with direct command and online commands.

7 : Commands which can be executed only when an arithmetic processor is provided (also includes condition 6).

◯ ***The memory usage parameter displays the status of the object memory and not the variable usage amount.***
If one numerical variable is used, 4 bytes of memory are occupied.
If one character string variable is used, 84 bytes of memory are occupied.

EXAMPLES:

A = 1

------ 13+4=17 bytes of memory are occupied.

A = 1 0

------ 13 bytes of memory are reserved for A.

◯ ***When a DIM statement is used to define an array, array variable space is reserved.***
If DIM X(l, m, n) is defined, (l+1)*(m+1)*(n+1)*4 bytes of object memory are occupied.

EXAMPLE:

D I M A (2 , 3 , 1)

------ (2+1)(3+1)*(1+1)*4=96 bytes are occupied.*

DIM statement memory occupancy is 14 bytes, so the total amount is 110 bytes.

15 Robot Language Function List

15-1 Arithmetical Functions

Types of Functions and Variables	Format	Example	Memory Usage [bytes]
DI	(1)DIm([b, ..., b]) (2)DI(mb, ..., mb)	A=DI0( ) A=DI2(7, 5, 0) A=DI(37, 35, 25, 20)	13 15 24
DO	(1)DOm([b, ..., b]) (2)DO(mb, ..., mb)	DO2( )=&B10101010 DO(37, 35, 23, 20)=&H0C	13 24
MO	(1)MOm([b, ..., b]) (2)MO(mb, ..., mb)	MO2()=&B10101010 MO(37, 35, 23, 20)=&H0C	13 24
TO	(1)TOm([b, ..., b]) (2)TO(mb, ..., mb)	TO0()=&B10101010 TO(07, 05, 03, 00)=&H0C	13 18
LO	(1)LOm([b, ..., b]) (2)LO(mb, ..., mb)	LO0()=&B10101010 LO(07, 05, 03, 00)=&H0C	13 18
Array Variables	variable(<expression>, [<expression>, [<expression>]])	A(0)=12345.67 B(0, 2, 1)=A(0)	20 31
Point Element Variables	LOCx(<point expression>)	LOCX(P0)=123.45 LOCY(P0)=LOCY(P[A])+LOCY(P[B])	18 30
Shift Element Variable	LOCx(<shift expression>)	LOCX(S0)=100.00 LOCR(S[A])=LOCR(S9)	17 20
ABS	ABS(<expression>)	A(0)=ABS(-123.45) A(1)=ABS(B/2+C)	22 27
ACCEL	ACCEL(<expression>)	IF ACCEL(1)=50 THEN *ABC	22
ACCEL2	ACCEL2(<expression>)	LOCX(P[B])=ACCEL2(1)	19
ARCH	ARCH(<expression>)	IF ARCH(1)=500 THEN *ABC	22
ARCH2	ARCH2(<expression>)	B=ARCH2(I%)	14
ATN	ATN(<expression>)	A(0)=ATN(0.5) A(1)=ATN(Y/X)	21 23
AXWGHT	AXWGHT(<expression>)	B%(D%)=AXWGHT(D%)	19
AXWGHT2	AXWGHT2(<expression>)	B%(D%)=AXWGHT2(D%)	19
COS	COS(<expression>)	A(0)=COS(DEGRAD (45)) A(1)=B*COS(C)	20 23
DEGRAD	DEGRAD(<expression>)	A=COS(DEGRAD(30))	15
DIST	DIST(<point expression 1>), <point expression 2>)	A=DIST(P0,P1)	19
ERR	ERR	IF ERR<>&H604 THEN HALT	17
ERL	ERL	IF ERL=20 THEN RESUME NEXT	18
INT	INT(<expression>)	B=INT(-1.233)	17
LEN	LEN(<character string expression>)	B=LEN(A$)	15

Types of Functions and Variables	Format	Example	Memory Usage [bytes]
LSHIFT	LSHIFT(<expression>, <expression>)	A=LSHIFT(B(0), 2)	22
ORD	ORD(<character string expression>)	A=ORD("B")	17
OUTPOS	OUTPOS(<expression>)	IF OUTPOS(1)=500 THEN * ABC	22
OUTPOS2	OUTPOS2(<expression>)	A(B)=OUTPOS2(B)	19
RADDEG	RADDEG(<expression>)	A=RADDEG(ATN(B))	19
RSHIFT	RSHIFT(<expression>, <expression>)	A=RSHIFT(B(0), 2)	22
SIN	SIN(<expression>)	A(0)=SIN(DEGRAD (60)) A(1)=SIN(B * 2+C)	20 27
SQR	SQR(<expression>)	A(0)=SQR(4) A(1)=SQR(B+C+D)	19 27
TAN	TAN(<expression>)	A(0)=TAN(3.141592 * 45/180) A(1)=B+TAN(C + 45)	29 23
TIMER	TIMER	A%=TIMER	11
TOLE	TOLE(<expression>)	IF TOLE(1)=10 THEN * ABC	22
TOLE2	TOLE2(<expression>)	A%=TOLE2(1)	14
VAL	VAL(<character string expression>)	A=VAL("&B10001")	23
WEIGHT	WEIGHT	IF WEIGHT=0 THEN * ABC	19
WEIGHT2	WEIGHT2	IF WEIGHT2=0 THEN HALT "ERROR"	26

15-2 Character String Functions

Types of Functions and Variables	Format	Example	Memory Usage [bytes]
CHR$	CHR$(<expression>)	A$=CHR$(A)	15
DATE$	DATE$	PRINT DATE$	6
LEFT$	LEFT$(<character string expression>, <expression>)	C$=LEFT$(A$,4)	19
MID$	MID$(<character string expression>, <expression> [, <expression>]	D$=MID$(A$, 4, 2)	23
RIGHT$	RIGHT$(<character string expression>, <expression>)	B$=RIGHT$(A$, 4)	19
STR$	STR$(<expression>)	A$=STR$(10.01)	17
TIME$	TIME$	PRINT TIME$	6

15-3 Point Functions

Types of Functions and Variables	Format	Example	Memory Usage [bytes]
JTOXY	JTOXY(<point expression>)	P10=JTOXY(WHERE)	12
JTOXY2	JTOXY2(<point expression>)	P10=JTOXY2(WHERE2)	12
WHERE	WHERE	P0=WHERE	11
WHERE2	WHERE2	P0=WHERE2	11
XYTOJ	XYTOJ(<point expression>)	P10=XYTOJ(P10)	14
XYTOJ2	XYTOJ2(<point expression>)	P10=XYTOJ2(P10)	14

POINT

Refer to "13 Command Statement List" for more details on the memory usage.

16 Data File Details

The following is an explanation of the data files used with the SEND command.

POINT

- *All data is handled as character strings of ASCII character codes.*
- *No lower case alphabetic characters may be used.*
- *See the table of character codes for more details.*

There are 24 different types of data files:

1. Program Files	*2. Point Data Files*
3. Parameter Files	*4. Shift Data Files*
5. Hand Data Files	*6. System Files*
7. Palette Definition Files	*8. Variable Files*
9. Array Variable Files	*10. Character Strings*
11. Directory Files	*12. Free Memory Status*
13. Point Data Use Files	*14. DI Files*
15. DO Files	*16. MO Files*
17. LO Files	*18. TO Files*
19. DIO Files	*20. Communication Files*
21. Console Input Files	*22. Console Output Files*
23. Machine Reference Files	*24. EOF Files*

16-1 Program Files

These files specify user programs for the robot. There are 2 types; one specifies all the programs, and the other specifies one program.

All programs

FORMAT:

```
P G M
```

- ◯ *Expresses all programs.*
- ◯ *When used for a read-out file, all programs that are currently registered are read out.*
- ◯ *When used for a write-in file, a program name is necessary.*

DATA FORMAT:

```
NAME=program namec/r
□□□□□□□□□□□□•••••□□□□□□□□c/r
□□□□□□□□□□□□•••••□□□□□□□□c/r
          •
          •
□□□□□□□□□□□□•••••□□□□□□□□c/r
NAME=program namec/r
          •
          •
NAME=program namec/r
□□□□□□□□□□□□•••••□□□□□□□□c/r
□□□□□□□□□□□□•••••□□□□□□□□c/r
          •
          •
□□□□□□□□□□□□•••••□□□□□□□□c/r
c/r
```

- ◯ *Program names consist of 8 alphanumeric characters or the underscored ("_") characters.*
- ◯ *□: indicates a character code. (TAB code cannot be used.)*
- ◯ *c/r shows CR code (0Dh)+LF code (0Ah).*
- ◯ *Add a c/r alone signifies the end of the file at the end of file.*
- ◯ *A line must consist of 75 characters or less.*

Example:

```
SEND PGM TO CMU
        ---------- Outputs all programs to the communication port
                    as follows.
SEND CMU TO PGM
        ---------- Inputs all programs from the communication port
                    as follows.

NAME=UPDOWNc/r
*ST1: c/r
      RESET DO2() c/r
                :
                :
NAME=LEFTc/r
                :
                :
NAME=RIGHTc/r
*STN: c/r
      MOVE L, P0, Z=0c/r
                :
                :
          HALTc/r
          c/r
```

One program

FORMAT:

<program name>

- *Expresses one program.*
- *Program names consist of 8 alphanumeric characters or the underscored ("_") character, surrounded by "<" and ">".*
- *If "<>" is specified without any program name inside, the program currently called up will be specified.*

Example:

```
<ABC>,<>
```

DATA FORMAT:

```
NAME=program namec/r
□□□□□□□□□□□□•••••□□□□□□□□c/r
□□□□□□□□□□□□•••••□□□□□□□□c/r
          •
          •
□□□□□□□□□□□□•••••□□□□□□□□c/r
c/r
```

- ◯ *Program names consist of 8 alphanumeric characters or the underscored ("_") characters.*
- ◯ *□: indicates a character code. (TAB code cannot be used.)*
- ◯ *c/r shows CR code (0Dh)+LF code (0Ah).*
- ◯ *Add a c/r alone signifies the end of the file at the end of file.*
- ◯ *A line must consist of 75 characters or less.*

Example:

```
SEND <UPDOWN> TO CMU
        ---------- Outputs program "UPDOWN" to communication
                   port as follows.
SEND CMU TO <ABC>
        ---------- Inputs program from communication port and
                   changes the program name to "ABC" as follows.

NAME=UPDOWNc/r
*ST1: c/r
    RESET DO2 () c/r
          :
    MOVE L, P0, Z=0 c/r
          :
          :
HALTc/r
c/r
```

CAUTION

- ***When the current mode is "AUTO" or "PROGRAM" mode and writing is attempted to the currently specified program name, an "OVERWRITE PROHIBITED" error message is displayed and the SEND command is halted.***
- ***When a sequence program is executed and the program name is "SEQUENCE", an "OVERWRITE PROHIBITED" error message is displayed and the SEND command is halted.***
- ***When writing into files, an error message "Program doesn't exist" will result if there is no NAME statement in the program. The SEND command will be stopped.***
- ***Program names in SEND command are surrounded by "<" and ">". However "<" and ">" are not needed for program names in the file.***

16-2 Point Data Files

These files hold the point data for the robot. There are 4 types of point expressions: all points, one points, expression specified one point, and multiple points expressed with hyphens.

All points

FORMAT:

```
P N T
```

- *Expresses all point data.*
- *When used for a read-out file, all currently stored point data is read out.*

DATA FORMAT:

```
Pm=fnnnnnn_fnnnnnn_fnnnnnn_fnnnnnn_fnnnnnn_fnnnnnnc/r
Pm=fnnnnnn_fnnnnnn_fnnnnnn_fnnnnnn_fnnnnnn_fnnnnnnc/r
                    •
                    •
Pm=fnnnnnn_fnnnnnn_fnnnnnn_fnnnnnn_fnnnnnn_fnnnnnnc/r
c/r
```

- *m is 0 to 4000*
- *f is shown by a +, - or space.*
- *nnnnnn is shown by a numeric value within 6 digits.*
- *_ is a space code.*
- *c/r shows CR code (0Dh)+LF code (0Ah).*
- *Add a c/r alone signifies the end of the file at the end of file.*
- *A line must consist of 75 characters or less.*

Example:

```
S E N D  P N T  T O  C M U
        ----------------- Outputs all points to the communication port as
                          follows.
S E N D  C M U  T O  P N T
        ----------------- Inputs all points from the communication port as
                          follows.

P0    =      0       0       0       0       0      0c/r
P10   =   1000    1000    1000    1000       0      0c/r
        :
P155  =  155.20  -250.0  150.00 -180.00    0.00   0.00c/r
c/r
```

One point

FORMAT:

P m (m=0 to 4000)

- *m is 0 to 4000*
- *f is shown with a +, a - or a space.*
- *nnnnnn is shown by a numeric value within 6 digits.*
- *_ is a space code.*
- *c/r shows CR code (0Dh)+LF code (0Ah).*
- *A line must consist of 75 characters or less.*
- *Expresses one specified point.*

DATA FORMAT:

Pm=fnnnnnn_fnnnnnn_fnnnnnn_fnnnnnn_fnnnnnn_fnnnnnnc/r

Example:

S E N D C M U T O P 1 0 0

---------- Inputs one point from the communication port as follows.

P10 = 1000 1000 1000 1000 0 0c/r

One point specified by an <expression>

Format:

P<expression>

- *Expresses one point specified by an <expression>.*

DATA FORMAT:

Pm=fnnnnnn_fnnnnnn_fnnnnnn_fnnnnnn_fnnnnnn_fnnnnnnc/r

- *m is 0 to 4000*
- *f is shown with a +, a - or a space.*
- *nnnnnn is shown by a numeric value within 6 digits.*
- *_ is a space code.*
- *c/r shows CR code (0Dh)+LF code (0Ah).*
- *A line must consist of 75 characters or less.*

Example:

S E N D P [A] T O C M U

---------- Outputs P100 to the communication port if A is 100 as follows.

P100 = 0 0 0 0 0 0c/r

Multiple points expressed with hyphens

FORMAT:

```
P – – – –
```

◯ *Expresses many points specified by an <expression>.*

Example:

P 1 0 –
---------- 10 points of P100 to P109
P 1 – 2
---------- 10 points of P102, P112, •••, P192
P – – 5
---------- 100 points of P5, P15, •••, P985, P995

DATA FORMAT:

```
Pm=fnnnnnn_fnnnnnn_fnnnnnn_fnnnnnn_fnnnnnn_fnnnnnnc/r
Pm=fnnnnnn_fnnnnnn_fnnnnnn_fnnnnnn_fnnnnnn_fnnnnnnc/r
          •
          •
Pm=fnnnnnn_fnnnnnn_fnnnnnn_fnnnnnn_fnnnnnn_fnnnnnnc/r
c/r
```

◯ *m is 0 to 4000*
◯ *f is shown with a +, a - or a space.*
◯ *nnnnnn is shown by a numeric value within 6 digits.*
◯ *_ is a space code.*
◯ *c/r shows CR code (0Dh)+LF code (0Ah).*
◯ *A line must consist of 75 characters or less.*

Example:

S E N D P 1 0 – 0 T O C M U
---------- Outputs points such as below to the communication port.

```
P1000 =   1000    1000    1000    1000      0    0c/r
P1010 =   2000    2000    2000    2000      0    0c/r
   :
P1090 =  10000   10000   10000   10000      0    0c/r
c/r
```

CAUTION

Point data contents are expressed in integral values with pulse units, and in real values with millimeters.

16-3 Parameter Files

The parameter file contains all the parameters for the robot.

FORMAT:

```
P R M
```

- ◯ *Expresses all parameters.*
- ◯ *When used for a read-out file, all current parameter settings are read out.*
- ◯ *When used for a write-in file, only parameters specified with a parameter label are read out.*

DATA FORMAT:

```
¥ parameter label ¥ '<comment>c/r
<robot data>c/r
¥ parameter label ¥ '<comment>c/r
<Data classified by axis>c/r
          :
¥ parameter label ¥ '<comment>c/r
<Robot data>c/r
¥ parameter label ¥ '<comment>c/r
<Data classified by axis>c/r
          :
c/r
```

- ◯ *Parameter labels are shown with 6 alphabetic characters.*
- ◯ *The "/" at the front and back of the parameter label indicates an open parameter and the "\" indicates a closed parameter (without a description in the manual). The closed parameter cannot be checked on the MPB.*
- ◯ *<comment> is shown in Japanese or English according to the display character on the MPB.*
- ◯ *c/r shows CR code (0Dh)+LF code (0Ah).*
- ◯ *Add a c/r alone signifies the end of the file at the end of file.*
- ◯ *A line must consist of 75 characters or less.*

Example:

```
S E N D   P R M   T O   C M U
        ---------- Outputs all parameters to the communication port
                   as follows.

\RBTSPC\ 'ROBOT SPEC c/r
 R1= 5137 c/r
        :
\DATLEN/ 'Data display length c/r
 RC = 0 c/r
/ACCEL / 'Accel coefficient[%] c/r
 D1=  100  D2= 100  D3= 100  D4= 100  c/r
        :
\NMOTOR\ 'Motor type c/r
 D1=  196  D2= 196  D3= 196  D4= 195  c/r
c/r

S E N D   C M U   T O   P R M
        ---------- nputs + software limit from the communication port
                   as follows.

/PLMT+/'+Soft limit[pulse] c/r
D1= 487424   D2= 440548  D3= 327680   D4= 409600  c/r
c/r
```

POINT

Refer to the separate "12-1 Parameter" in Chapter 4 in the Operation Manual for the meanings of parameters.

CAUTION

Parameters are verified to be compatible with a upper version. However, compatibility with a lower version may not be maintained.

16-4 Shift Data Files

Shift data files hold the shift coordinate data for the robot. This data can be expressed in three methods: all 10 sets of shift data, one set of shift data or one set of shift data by <expression>.

All shift data

FORMAT:

```
S F T
```

- *Expresses all shift coordinate data.*

DATA FORMAT:

```
Sm    =fnnnnnn_fnnnnnn_fnnnnnn_fnnnnnnc/r
SPm   =fnnnnnn_fnnnnnn_fnnnnnn_fnnnnnnc/r
SMm   =fnnnnnn_fnnnnnn_fnnnnnn_fnnnnnnc/r
                    :
SPm   =fnnnnnn_fnnnnnn_fnnnnnn_fnnnnnnc/r
SMm   =fnnnnnn_fnnnnnn_fnnnnnn_fnnnnnnc/r
c/r
```

- *m is 0 to 9.*
- *f is shown with a +, a - or a space.*
- *nnnnnn is show by a real number within 2 digits of the decimal point.*
- *_ is the space code.*
- *c/r shows CR code (0Dh)+LF code (0Ah).*
- *Add a c/r alone signifies the end of the file at the end of file.*
- *A line must consist of 75 characters or less.*

Example:

S E N D S F T T O C M U

---------- Outputs all shift data to the communication port as follows.

S E N D C M U T O S F T

---------- Inputs all shift data from the communication port as follows.

S0	*=*	*0.00*	*0.00*	*0.00*	*0.00c/r*
SP0	*=*	*0.00*	*0.00*	*0.00*	*0.00c/r*
SM0	*=*	*0.00*	*0.00*	*0.00*	*0.00c/r*
S1	*=*	*10.00*	*-20.00*	*90.00*	*50.00c/r*
SP1	*=*	*0.00*	*0.00*	*0.00*	*0.00c/r*
SM1	*=*	*0.00*	*0.00*	*0.00*	*0.00c/r*
	:				
S9	*=*	*0.00*	*0.00*	*0.00*	*0.00c/r*
SP9	*=*	*0.00*	*0.00*	*0.00*	*0.00c/r*
SM9	*=*	*-155.20*	*-250.01*	*-50.00*	*-180.00c/r*
c/r					

One set of shift

FORMAT:

```
S m  (m=0 to 9)
```

- *Expresses the specified one set of shift data.*

DATA FORMAT:

```
Sm   =fnnnnnn_fnnnnnn_fnnnnnn_fnnnnnnc/r
SPm  =fnnnnnn_fnnnnnn_fnnnnnn_fnnnnnnc/r
SMm =fnnnnnn_fnnnnnn_fnnnnnn_fnnnnnnc/r
c/r
```

- *m is 0 to 9.*
- *f is shown with a +, a - or a space.*
- *nnnnnn is show by a real number within 2 digits of the decimal point.*
- *_ is space code.*
- *c/r shows CR code (0Dh)+LF code (0Ah).*
- *Add a c/r alone signifies the end of the file at the end of file.*
- *A line must consist of 75 characters or less.*

Example:

S E N D C M U T O S 0

---------- Inputs shift data from the communication port as follows.

```
S0    =     10.00      -20.00      90.00     50.00c/r
SP0   =    999.00      400.00     180.00    100.00c/r
SM0   =     -5.00      -20.00    -180.00      0.00c/r
c/r
```

One set of shift specified by <expression>

FORMAT:

```
S [<expression>]
```

- *m is 0 to 9.*
- *f is shown with a +, a - or a space.*
- *nnnnnn is show by a real number within 2 digits of the decimal point.*
- *_ is space code.*
- *c/r shows CR code (0Dh)+LF code (0Ah).*
- *Add a c/r alone signifies the end of the file at the end of file.*
- *A line must consist of 75 characters or less.*
- *Expresses one set of shift specified by <expression>.*

DATA FORMAT:

```
Sm   =fnnnnnn_fnnnnnn_fnnnnnn_fnnnnnnc/r
SPm  =fnnnnnn_fnnnnnn_fnnnnnn_fnnnnnnc/r
SMm =fnnnnnn_fnnnnnn_fnnnnnn_fnnnnnnc/r
c/r
```

Example:

S E N D S [A] T O C M U

---------- Outputs S1 to communication port if A is 1 as follows.

```
S1    =     10.00      -20.00      90.00     50.00c/r
SP1   =    999.00      400.00     180.00    100.00c/r
SM1   =     -5.00      -20.00    -180.00      0.00c/r
c/r
```

16-5 Hand Data Files

These files hold all of the Hand data for the robot. There are two methods to express hand data: all 8 hand data files (4 sets for the main robot and 4 sets for the sub robot), or one set of hand data.

All hand data

FORMAT:

```
H N D
```

- ◯ *Expresses all hand data.*

DATA FORMAT:

```
Hm=fnnnnnn_fnnnnnn_fnnnnnn_[R]c/r
          :
Hm=fnnnnnn_fnnnnnn_fnnnnnn_[R]c/r
c/r
```

- ◯ *m is 0 to 7.*
- ◯ *f is shown with a +, a - or a space.*
- ◯ *nnnnnn is shown by a real number within 2 digits of the decimal point or by an integer within 6 digits.*
- ◯ *_ is space code.*
- ◯ *[R] shows whether an R-axis is installed.*
- ◯ *c/r shows CR code (0Dh)+LF code (0Ah).*
- ◯ *Add a c/r alone signifies the end of the file at the end of file.*
- ◯ *A line must consist of 75 characters or less.*

Example:

```
S E N D  H N D  T O  C M U
        ---------- Outputs all hand data to the communication port
                   as follows.
S E N D  C M U  T O  H N D
        ---------- Inputs all hand data from the communication port
                   as follows.

H0   =    10.00     10.00     30.00     Rc/r
H1   =        0      0.00      0.00      c/r
     :
H7   =     0.00      0.00      0.00      c/r
c/r
```

POINT

H0 to H3 is used for the main robot, and H4 to H7 is used for the sub robot.

One hand data

FORMAT:

H m (m=0 to 7)

- ◯ *Expresses the specified one set of hand.*

DATA FORMAT:

Hm=fnnnnnn_fnnnnnn_fnnnnnn_[R] c/r

- ◯ *m is 0 to 7.*
- ◯ *f is shown with a +, a - or a space.*
- ◯ *nnnnnn is shown by a real number within 2 digits of the decimal point or by an integer within 6 digits.*
- ◯ *_ is space code.*
- ◯ *[R] shows whether an R-axis is installed.*
- ◯ *c/r shows CR code (0Dh)+LF code (0Ah).*
- ◯ *Add a c/r alone signifies the end of the file at the end of file.*
- ◯ *A line must consist of 75 characters or less.*

Example:

```
SEND CMU TO H0
        ---------- Inputs hand data from the communication port as
                   follows.

H0   =    10.00     10.00     30.00     Rc/r
```

16-6 System Files

This file handles the robot user memory in one batch.

FORMAT:

```
ALL
```

DATA FORMAT:

```
PGM  File
PNT  File
PRM  File
SFT  File
HND  File
```

Example:

SEND ALL TO CMU

---------- Outputs a system to the communication port. This expression is equivalent to executing all at once as follows.

```
SEND PGM TO CMU
SEND PNT TO CMU
SEND PRM TO CMU
SEND SFT TO CMU
SEND HND TO CMU
```

SEND CMU TO ALL

---------- Inputs a system from the communication port. This expression is equivalent to executing all at once as follows.

```
SEND CMU TO PGM
SEND CMU TO PNT
SEND CMU TO PRM
SEND CMU TO SFT
SEND CMU TO HND
```

POINT

Refer to the file description for details on each file.

16-7 Palette Definition Files

This file indicates palette defined data for the robot. There are two methods, one shows all 10 sets of palette definition data, and the other shows one set of palette definition data.

All palette definition

FORMAT:

```
P L T
```

- *Expresses all palette definition data.*

DATA FORMAT:

```
PLmc/r
PLN = XYc/r
NX =kkkc/r
NY =kkkc/r
NZ =kkkc/r
P[1] =fnnnnnn_fnnnnnn_fnnnnnn_fnnnnnn_fnnnnnn_fnnnnnnc/r
P[2] =fnnnnnn_fnnnnnn_fnnnnnn_fnnnnnn_fnnnnnn_fnnnnnnc/r
P[3] =fnnnnnn_fnnnnnn_fnnnnnn_fnnnnnn_fnnnnnn_fnnnnnnc/r
P[4] =fnnnnnn_fnnnnnn_fnnnnnn_fnnnnnn_fnnnnnn_fnnnnnnc/r
P[5] =fnnnnnn_fnnnnnn_fnnnnnn_fnnnnnn_fnnnnnn_fnnnnnnc/r
          :
PLmc/r
          :
P[5] =fnnnnnn_fnnnnnn_fnnnnnn_fnnnnnn_fnnnnnn_fnnnnnnc/r
c/r
```

- *m is 0 to 9.*
- *k is a positive integer.*
- *f is shown with a +, a - or a space.*
- *nnnnnn shows by real number within 2 digits.*
- *_ is space code.*
- *c/r shows CR code (0Dh)+LF code (0Ah).*
- *Add a c/r alone signifies the end of the file at the end of file.*
- *A line must consist of 75 characters or less.*

Example:

S E N D P L T T O C M U

---------- Outputs all palette definition data to the communication port as follows.

```
PLOc/r
PLN   =   XYc/r
NX    =    3c/r
NY    =    4c/r
NZ    =    2c/r
P[1]  =    0.00     0.00     0.00     0.00     0.00   0.00c/r
P[2]  =  100.00     0.00     0.00     0.00     0.00   0.00c/r
P[3]  =    0.00   100.00     0.00     0.00     0.00   0.00c/r
P[4]  =  100.00   100.00     0.00     0.00     0.00   0.00c/r
P[5]  =    0.00     0.00    50.00     0.00     0.00   0.00c/r
       :
PL4c/r
PLN   =   XYc/r
       :
P[5]  =    0.00     0.00     0.00     0.00     0.00   0.00c/r
c/r
```

One set of palette definition

FORMAT:

P L m (m=0 to 9)

○ *Expresses one set of palette definition data.*

DATA FORMAT:

```
PLmc/r
PLN = XYc/r
NX  =kkkc/r
NY  =kkkc/r
NZ  =kkkc/r
P[1] =fnnnnnn_fnnnnnn_fnnnnnn_fnnnnnn_fnnnnnn_fnnnnnnc/r
P[2] =fnnnnnn_fnnnnnn_fnnnnnn_fnnnnnn_fnnnnnn_fnnnnnnc/r
P[3] =fnnnnnn_fnnnnnn_fnnnnnn_fnnnnnn_fnnnnnn_fnnnnnnc/r
P[4] =fnnnnnn_fnnnnnn_fnnnnnn_fnnnnnn_fnnnnnn_fnnnnnnc/r
P[5] =fnnnnnn_fnnnnnn_fnnnnnn_fnnnnnn_fnnnnnn_fnnnnnnc/r
c/r
```

- *m is 0 to 9.*
- *k is a positive integer.*
- *f is shown with a +, a - or a space.*
- *nnnnnn shows by real number within 2 digits.*
- *_ is space code.*
- *c/r shows CR code (0Dh)+LF code (0Ah).*
- *Add a c/r alone signifies the end of the file at the end of file.*
- *A line must consist of 75 characters or less.*

Example:

S E N D C M U T O P L 0

---------- Inputs palette definition data from the communication port as follows.

```
PL0c/r
PLN  =   XYc/r
NX   =    3c/r
NY   =    4c/r
NZ   =    2c/r
P[1] =    0.00    0.00    0.00    0.00    0.00   0.00c/r
P[2] =  100.00  100.00    0.00    0.00    0.00   0.00c/r
P[3] =    0.00  200.00    0.00    0.00    0.00   0.00c/r
P[4] =  100.00  200.00    0.00    0.00    0.00   0.00c/r
P[5] =    0.00    0.00  100.00    0.00    0.00   0.00c/r
c/r
```

CAUTION

Point data in the palette definition uses the data area as follows.
QRC, QRCH, MRC (with extension RAM), MRCH series
Palette definition 0: P[1] to P[5]=P3996 to P4000
Palette definition 1: P[1] to P[5]=P3991 to P3995
:
Palette definition 9: P[1] to P[5]=P3951 to P3955
MRC (without extension RAM) series
Palette definition 0: P[1] to P[5]=P1596 to P1600
Palette definition 1: P[1] to P[5]=P1591 to P1595
:
Palette definition 9: P[1]to P[5]=P1551 to P1555

16-8 Variable Files

The variable file contains variables used by the robot. Only global variables are contained. The variable which is defined in SUB statement is not handled by this file. Two methods are used here, one indicates all variables, the other indicates one variable.

All variables

FORMAT:

```
V A R
```

- ◯ *Expresses all variables.*
- ◯ *All global variable data is read out when used as a read-out file.*
- ◯ *Specified variables are written into, when used as a writing file.*

DATA FORMAT:

```
<variable name>x=<constant>c/r
<variable name>x=<constant>c/r
          •
          •
<variable name>x=<constant>c/r
c/r
```

- ◯ *The <variable name> consists of alphanumeric characters at the beginning of character string and is shown with up to 16 alphanumeric characters and underscored (_) characters.*
- ◯ *x is integer type: %, real number type: !, character type: $.*
- ◯ *<constant> is shown as follows according to the type of variable.*
  Integer type: nnnnnnn nnnnnnn is an integral value of 7 digits or less.
  Real number type: mmmmmmmm mmmmmmmm is integer part + fractional part of 7 digits or less.
  Character type: "ssssss•••sss" ss••ss is a character string of 70 character or less.
- ◯ *Add a c/r alone signifies the end of the file at the end of file.*
- ◯ *A line must consist of 75 characters or less.*

Example:

S E N D V A R T O C M U

---------- Outputs all variables to the communication port as follows.

S E N D C M U T O V A R

---------- Inputs all variable from the communication port as follows.

```
SGI0=0c/r
      :
SGI7=0c/r
SGR0=0.0c/r
      :
SGR7=0.0c/r
A!=    1.12E2c/r
B!=   10000.5c/r
N%=       4c/r
Z%=      -2c/r
WS1$=    "0.00    0.00    0.00    0.00 0.00"c/r
ZS2$= " YAMAHA ROBOT "c/r
c/r
```

CAUTION

SGIx indicates real number type static global variables and SGRx indicates integer type static global variables.

One variable

FORMAT:

```
a b b b b b b b b b b b b b b x
```

- *Expresses the specified one variable.*
- *a is an alphabetic character, b is an alphanumeric character or underscored (_) of up to 16 characters.*
- *x=%, !, $*

DATA FORMAT:

Integer type:

```
k k k k k k k c / r
```

- *kkkkkkk is an integer within 7 digits.*
- *c/r shows CR code (0Dh)+LF code (0Ah).*

Real number type:

```
nnnnnnnnc/r
```

- *nnnnnnnn is integer part + fractional part by a real number within 7 digits.*
- *c/r shows CR code (0Dh)+LF code (0Ah).*

Character type:

```
ssssss...sssssc/r
```

- *sss...sss is a character string within 70 characters.*
- *c/r shows CR code (0Dh)+LF code (0Ah).*
- *A line must consist of 75 characters or less.*

Example:

```
SEND A! TO CMU
        ---------- Outputs variable A! to the communication port as
                   follows.
SEND CMU TO A!
        ---------- Inputs variable A! from the communication port as
                   follows.

1.12E2c/r

SEND CMU TO S$
        ---------- Inputs variable S$ from the communication port
                   as follows.
SEND S$ TO CMU
        ---------- Outputs variable S$ to the communication port as
                   follows.

THIS IS XY ROBOT OF YAMAHA MOTORc/r
```

CAUTION

Only variables that have previously been entered (registered) are used during compiling. When variables that have not been entered (registered) are used, the error message "Undefined identifier" appears.

16-9 Array Variable Files

The variable file contains array variables used by the robot. Only global array variables are contained. The array variable which is defined in SUB statement is not handled by this file. Two methods are used here, one indicates all array variables, the other indicates one array variable.

All array variables

FORMAT:

```
A R Y
```

- ◯ *Expresses all array variables.*
- ◯ *All global array variable data is read out when used as a read-out file.*
- ◯ *Specified array variables are written into, when used as a writing file.*

DATA FORMAT:

```
<array variable name>x(k)=<constant>c/r
<array variable name>x(k)=<constant>c/r
            •
            •
<array variable name>x(k)=<constant>c/r
c/r
```

- ◯ *The <array variable name> consists of alphanumeric characters at the beginning of character string and is shown with up to 16 alphanumeric characters and underscored (_) characters.*
- ◯ *x is integer type: %, real number type: !, character type: $.*
- ◯ *k is a positive integer.*
- ◯ *<constant> is shown as follows according to the type of variable.*
  Integer type: nnnnnnn nnnnnnn is an integral value of 7 digits or less.
  Real number type: mmmmmmmm mmmmmmmm is integer part + fractional part of 7 digits or less.
  Character type: "ssssss•••sss" ss••ss is a character string of 70 character or less.
- ◯ *Add a c/r alone signifies the end of the file at the end of file.*
- ◯ *A line must consist of 75 characters or less.*

Example:

S E N D A R Y T O C M U

---------- Outputs all variables to the communication port as follows.

```
A!(0)=  1.12E2c/r
A!(1)=  1.12E2c/r
B!(0)=   0.0E0c/r
B!(1)=   1.1E1c/r
B!(2)=   1.2E1c/r
N%(0)=       4c/r
WS$(0)=    "0.00      0.00      0.00      0.00      0.00 0.00"c/r
WS$(1)=    "1.00      1.00      1.00      1.00      1.00 1.00"c/r
c/r
```

S E N D C M U T O A R Y

---------- Inputs all variable from the communication port as follows.

```
B!(2)= 10000.5c/r
N%(10)=      4c/r
Z%(11)=     -2c/r
WS$(2)=    "0.00      0.00      0.00      0.00      0.00 0.00"c/r
c/r
```

One array variable

FORMAT:

a b b b b b b b b b b b b b b b x (k)

- *Expresses the specified one variable.*
- *a is an alphabetic character, b is an alphanumeric character or underscored (_) of up to 16 characters.*
- *x=%, !, $*
- *k is a positive integer.*

DATA FORMAT:

Integer type:

k k k k k k k c / r

- *k is an integer within 7 digits.*
- *c/r shows CR code (0Dh)+LF code (0Ah).*

Real number type:

```
n n n n n n n n c / r
```

- ◯ *nnnnnnnn is integer part + fractional part by a real number within 7 digits.*
- ◯ *c/r shows CR code (0Dh)+LF code (0Ah).*

Character type:

```
s s s s s s . . . s s s s c / r
```

- ◯ *sss...sss is a character string within 70 characters.*
- ◯ *c/r shows CR code (0Dh)+LF code (0Ah).*
- ◯ *A line must consist of 75 characters or less.*

Example:

```
S E N D  A ! ( 2 )  T O  C M U
```

---------- Outputs array variable A!(2) to the communication port as follows.

```
1.12E2c/r
```

```
S E N D  C M U  T O  S $ ( 2 )
```

---------- Inputs array variable S$(2) from the communication port as follows.

```
T H I S  I S  X Y  R O B O T  O F  Y A M A H A  M O T O R c / r
```

CAUTION

Only array variables that have previously been entered (registered) are used during compiling. When variables that have not been entered (registered) are used, the error message "Undefined identifier" appears.

16-10 Character Strings

These files express character string data.
They are used only as read-out files.

FORMAT:

```
"<character string>"
```

- ○ *<character string> is 61 or less characters.*
- ○ *If you wish to include a double quotation mark in the string, use a pair of double quotation marks.*

DATA FORMAT:

```
□ □ . . . . . □ c/r
```

- ○ *□ is within 61 or less characters.*

Example:

```
SEND "THIS IS ""YAMAHA""" TO CMU
     ---------- Outputs a constant to the communication port as
                follows.

THIS IS "YAMAHA"c/r
```

16-11 Directory Files

These files hold user program data (program name, number of lines, number of bytes, type, date, time).
These are used only as read-out files.

All directory

FORMAT:

```
D I R
```

- ◯ *Expresses entire directory.*

DATA FORMAT:

```
No.Program    Lines Bytes Type Date Time c/r
nnnfgssssssss___kkkk_bbbbbb____xx______yy/mm/dd____hh:iic/r
                :
nnnfgssssssss___kkkk_bbbbbb____xx______yy/mm/dd____hh:iic/r
c/r
```

- ◯ *nnn is expressed with an integer of 3 digits or less.*
- ◯ *f is "o" when the name is same as the object, and f is "s" when the name is the same as the sequence object, others use a space.*
- ◯ *g is "*" when the current program is selected, others use a space.*
- ◯ *ssssssss is shown by 8 or less alphanumeric characters or by underscore (_).*
- ◯ *kkkk is shown by an integer of 4 digits or less.*
- ◯ *bbbbbb is shown by an integer of 6 digits or less.*
- ◯ *xx is RW (Readable and writable) or RO (Read only).*
- ◯ *yy, mm, dd respectively show the year, month, and day with two digits each.*
- ◯ *hh and ii respectively show the hour, and minutes with two digits each.*
- ◯ *_ shows the space code.*
- ◯ *c/r shows CR code (0Dh)+LF code (0Ah).*

Example:

S E N D D I R T O C M U

---------- Outputs entire directory to the communication port as follows.

No.	*Program*	*Lines*	*Bytes*	*Type*	*Date*	*Time c/r*
1	*TEST001*	*42*	*7174*	*RO*	*94/06/24*	*12:51*
*2o**	*TEST002*	*112*	*17623*	*RW*	*94/06/25*	*10:01*
			:			
6	*PTP*	*9*	*51*	*RW*	*94/07/21*	*13:12*
7s	*SEQUENCE ENDc/r*	*9*	*81*	*RW*	*95/01/21*	*15:21*

One directory

FORMAT:

```
<<program name>>
```

○ *expresses data of one program.*

DATA FORMAT:

```
No.Program    Lines BytesType Date Time c/r
nnnfgsssssss____kkkk_bbbbbb____xx_____yy/mm/dd____hh:iic/r
```

Example:

S E N D < < P T P > > T O C M U

---------- Outputs the program PTP data to the communication port as follows.

No.	*Program*	*Lines*	*Bytes*	*Type*	*Date*	*Time c/r*
6	*PTP*	*9*	*51*	*RW*	*94/07/21*	*13:12*

S E N D < < > > T O C M U

---------- Outputs the currently selected program data to the communication port as follows.

16-12 Free Memory Status

These files show the remaining available user memory (source program + point, object program).
These are used only as read-out files.

FORMAT:

```
M E M
```

- *Expresses entire memory.*

DATA FORMAT:

```
available bytes=nnnnnn/nnnnnnc/r
```

- *nnnnnn shows by integer 6 digits or less.*
- *c/r shows CR code (0Dh)+LF code (0Ah).*

Example:

```
S E N D  M E M  T O  C M U
        ---------- Outputs all memory to the communication port as
                   follows.

available bytes=90743/46380c/r
```

16-13 Point Data Use Files

These files express point usage status. This is helpful in finding out what point numbers are available.
These are used only as read-out files.

FORMAT:

```
S P N
```

DATA FORMAT:

```
P0-aaaaaaaaaaaaaaaaaaaaaaaaac/r
P25-aaaaaaaaaaaaaaaaaaaaaaaaac/r
                :
P3975-aaaaaaaaaaaaaaaaaaaaaaaaac/r
P4000-aaaaaaaaaaaaaaaaaaaaaaaaac/r
```

- *a is .= not used/1= used.*
  A single line shows 25 points.
- *c/r shows CR code (0Dh)+LF code (0Ah).*

Example:

```
S E N D  S P N  T O  C M U
        ---------- Outputs the status of point usage to the communi-
                    cation port as follows.

P0-1111111111.....11..1c/r
P25-.....1c/r
    :
P3975-1111.....c/r
P4000-.c/r
```

16-14 DI Files

These are the robot DI input files.
These are used only as read-out files.

FORMAT:

D I m ( ) m=0 to 7, 10 to 13

- *Expresses the DI input port.*

DATA FORMAT:

D I m ( ) = & B n n n n n n n n c / r

- *m is shown by integers from 0 to 7, 10 to 13.*
- *nnnnnnnn is shown by 8 digits consisting of 0 or 1. The DI input ports starting from the right are; m0, m1, .., m7.*
- *c/r shows CR code (0Dh)+LF code (0Ah).*

Example:

S E N D D I 2 ( ) T O C M U
---------- Outputs DI2() values to the communication port as follows.

D I 2 ( ) = & B 0 0 0 0 0 1 0 1 c / r
(DI(20) and DI(22) is ON)

16-15 DO Files

These are the robot DO output files.

FORMAT:

D O m ( ) m=0 to 7, 10 to 11

- ◯ *Expresses the DO output port.*

DATA FORMAT:
When a Read-out file:

D O m ( ) = & B n n n n n n n n c / r

When a write-in file:

& B n n n n n n n n c / r o r k c / r

- ◯ *m is shown by integers from 0 to 7, 10 to 11.*
- ◯ *nnnnnnnn is shown by 8 digits consisting of 0 or 1. The DO output ports starting from the right are; m0, m1, .., m7.*
- ◯ *k is shown by integer from 0 to 255.*
- ◯ *c/r shows CR code (0Dh)+LF code (0Ah).*

Example:

S E N D D O 2 ( ) T O C M U
---------- Outputs DO2() values to the communication port as follows.

D O 2 ( ) = & B 0 0 0 0 1 0 1 0 c / r
(DO(21) and DO(23) is ON)

S E N D C M U T O D O 2 ( )
---------- Inputs DO2() values from the communication port as follows.

& B 0 0 0 0 1 0 1 0 c / r o r 1 0 c / r
(DO(21) and DO(23) is ON)

CAUTION

DO0() and DO1() are read only.

16-16 MO Files

These are the robot MO output files.

FORMAT:

M O m ( ) m=0 to 7, 10 to 13

- ◯ *Expresses the MO output port.*

DATA FORMAT:
When a Read-out file:

M O m ( ) = & B n n n n n n n n c / r

When a write-in file:

& B n n n n n n n n c / r o r k c / r

- ◯ *m is shown by integers from 0 to 7, 10 to 13.*
- ◯ *nnnnnnnn is shown by 8 digits consisting of 0 or 1. The MO output ports starting from the right are; m0, m1, .., m7.*
- ◯ *k is shown by integer from 0 to 255.*
- ◯ *c/r shows CR code (0Dh)+LF code (0Ah).*

Example:

S E N D M O 2 ( ) T O C M U
---------- Outputs MO2() values to the communication port as follows.

M O 2 ( ) = & B 0 0 0 0 1 0 1 0 c / r
(MO(21) and MO(23) is ON)

S E N D C M U T O M O 2 ( )
---------- Inputs MO2() values from the communication port as follows.

& B 0 0 0 0 1 0 1 0 c / r o r 1 0 c / r
(MO(21) and MO(23) is ON)

CAUTION

MO0() and MO1() are read only.

16-17 LO Files

These are the robot LO output files.

FORMAT:

```
LO0()
```

- ◯ *Expresses the LO output port.*

DATA FORMAT:
When a Read-out file:

```
LOm()=&Bnnnnnnnnnc/r
```

When a write-in file:

```
&Bnnnnnnnnnc/r  or  kc/r
```

- ◯ *nnnnnnnn is 0 or 1 and 0, 1, .., 7 of LO output port from the right.*
- ◯ *k is shown by integer from 0 to 255.*
- ◯ *c/r shows CR code (0Dh)+LF code (0Ah).*

Example:

```
SEND  LO0()  TO  CMU
        ----------------- Outputs LO0() values to the communication port
                          as follows.

LO0()=&B00001010c/r
(LO(01) and LO(03) is ON)

SEND  CMU  TO  LO0()
        ----------------- Inputs LO0() values from the communication port
                          as follows.

&B00001010c/r  or  10c/r
(LO(01) and LO(03) is ON)
```

16-18 TO Files

These are the robot TO output files.

FORMAT:

```
TO0()
```

- ◯ *Expresses the TO output port.*

DATA FORMAT:
When a Read-out file:

```
TOm()=&Bnnnnnnnnc/r
```

When a write-in file:

```
&Bnnnnnnnnc/r  or  kc/r
```

- ◯ *nnnnnnnn is 0 or 1 and 0, 1, .., 7 of TO output port from the right.*
- ◯ *k is shown by integer from 0 to 255.*
- ◯ *c/r shows CR code (0Dh)+LF code (0Ah).*

Example:

```
SEND  TO0()  TO  CMU
        ----------------- Outputs TO0() values to the communication port
                          as follows.

TO0()=&B00001010c/r
(TO(01) and TO(03) is ON)

SEND  CMU  TO  TO0()
        ----------------- Inputs TO0() values from the communication port
                          as follows.

&B00001010c/r  or  10c/r
(TO(01) and TO(03) is ON)
```

16-19 DIO Files

These are the robot's DI input, DO output, MO output, LO output, TO output.

FORMAT:

```
D I O
```

○ *Expresses input/output port.*

DATA FORMAT:
Read-out file:

```
D I 0 ( ) = & B n n n n n n n n c / r
          :
D O 0 ( ) = & B n n n n n n n n c / r
          :
M O 0 ( ) = & B n n n n n n n n c / r
          :
L O 0 ( ) = & B n n n n n n n n c / r
T O 0 ( ) = & B n n n n n n n n c / r
```

○ *nnnnnnnn is 0 or 1 and 0, 1, .., 7 from the right.*

Example:

```
S E N D  D I O  T O  C M U
        ----------------- Outputs as follows.

D I 0 ( ) = & B 0 0 0 0 1 0 1 0 c / r
D I 1 ( ) = & B 1 1 0 0 0 1 1 0 c / r
     :
D O 0 ( ) = & B 0 1 0 0 1 1 1 1 c / r
     :
M O 0 ( ) = & B 1 1 1 1 1 1 1 1 c / r
     :
L O 0 ( ) = & B 0 0 0 0 1 0 1 0 c / r
T O 0 ( ) = & B 0 0 0 0 1 0 1 0 c / r
```

16-20 Communication Files

These are the robot communication files.

FORMAT:

```
C M U
```

DATA FORMAT:

This depends on the various data formats.
One line of data must be 75 characters or less.
When a read-out file, "^Z" (=1AH) is the file end code.

Example:

```
S E N D   P N T   T O   C M U
        ----------------- Outputs all points to the communication port as
                          follows.
S E N D   C M U   T O   P N T
        ----------------- Inputs all points from the communication port as
                          follows.

P0    =      0       0       0       0       0      0c/r
P10   =   1000    1000    1000    1000       0      0c/r
          :
P155  =  155.20 -250.01   50.00 -180.00    0.00   0.00c/r
c/r
```

16-21 Console Input Files

These are input files from the MPB (teaching box).
These are used only as read-out files.

FORMAT:

```
K E Y
```

DATA FORMAT:

This depends on the various data formats.
Input completion is performed with .

POINT

- ESC *key shows the end of a key file.*
- STOP *key stops execution.*

16-22 Console Output Files

These are output files to the MPB (teaching box).
These are used only as write-in files.

FORMAT:

```
SCR
```

DATA FORMAT:

This depends on the various data formats.
When a c/r is received the write position shifts to left end of the next line on the screen.

◯ *c/r shows CR code (0Dh)+LF code (0Ah).*

Example:

```
SEND CMU TO SCR
---------- Indicates the character string which is received as follows. ^Z(=1Ah) is necessary at the end of the data.

YAMAHA ROBOT  : SXYA_ARMc/r
CONTROLOR     : QRC24c/r
^Z
```

◯ *^Z shows EOF code (=1Ah).*

16-23 Machine Reference Files

This file shows the return to origin machine reference data of the robot. These are used only as read-out files.

FORMAT:

```
M R F
```

- *Expresses machine reference file.*

DATA FORMAT:

```
M 1 = n n n %M 2 = n n n %  . . .  n n n % c / r
S 1 = n n n %  S 2 = n n n %  . . .  n n n % c / r
c / r
```

- *nnn is integer of 0 to 100.*
- *c/r shows CR code (0Dh)+LF code (0Ah).*

Example:

```
S E N D  M R F  T O  C M U
        ----------------- Outputs machine reference to the communication
                          port as follows.
        ----------------- (when connecting to 4 axis robot)

M 1 =  5 4 %  M 2 =  4 4 %  M 3 = 5 6 %  M 4 = 5 2 % c / r
c / r
```

16-24 EOF Files

This is a custom file consisting only of the code: "^Z" (=1AH).
These are used only as read-out files.

FORMAT:

```
E O F
```

DATA FORMAT:

```
^ Z ( = 1 A h )
```

Example:

```
S E N D  P N T  T O  C M U
S E N D  E O F  T O  C M U
        ---------- Just as above, use when ^Z(=1Ah) is necessary at
                    the end of the data after the point file is output.

P0    =      0       0       0       0       0     0c/r
P10   =   1000    1000    1000    1000       0     0c/r
      :
P155  =  155.20 -250.01   50.00 -180.00    0.00  0.00c/r
c/r
^Z
```

○ *c/r shows CR code (0Dh)+LF code (0Ah).*

CAUTION

When using the communication port to send data to an external device, ^Z(=1Ah) is used to end the file.

17 User Program Examples

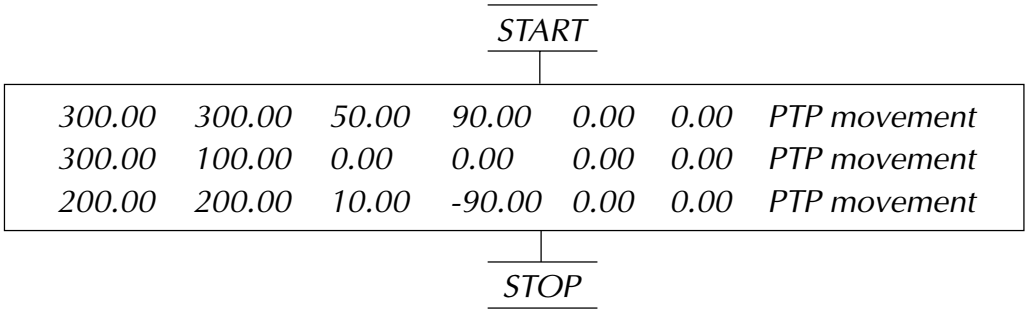
17-1 Basic Operation

17-1-1 Point Data Written Directly into Program

- *Outline*

  By inputting point data directly to the robot, the robot moves Point-To-Point (PTP). The shift coordinates are:ΔX=ΔY=ΔZ=Δtheta=0

- *Process Flow*



- *Program Example*

```
MOVE P, 300.00  300.00      50.00  90.00  0.00  0.00
MOVE P, 300.00  100.00      0.00   0.00   0.00  0.00
MOVE P, 200.00  200.00      10.00  -90.00 0.00  0.00
HALT
```

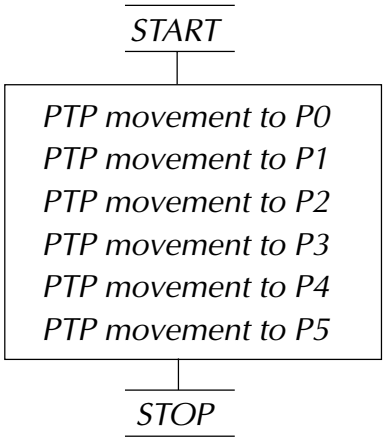
17-1-2 Using Point Numbers

- *Outline*

  During the program, a point number is used to specify coordinates. Coordinates are already input in "POINT" mode (MANUAL>POINT). Input the following data:

```
P0=   0.00    0.00    0.00    0.00    0.00    0.00
P1=   100.00  0.00    150.00          30.00   0.00    0.00
P2=   0.00    100.00          50.00   0.00    0.00    0.00
P3=   300.00  300.00          0.00    0.00    0.00    0.00
P4=   300.00  100.00          100.00          90.00   0.00    0.00
P5=   200.00  200.00          0.00    0.00    0.00    0.00
```

- *Process Flow*



- *Program Example*

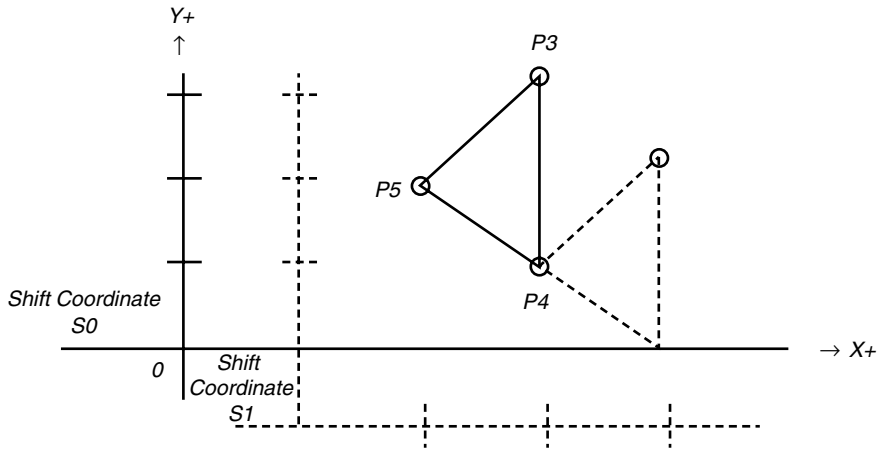
```
MOVE P, P0          FOR J=0 TO 5
MOVE P, P1          MOVE P, P[J]
MOVE P, P2          NEXT J
MOVE P, P3 or       HALT
MOVE P, P4
MOVE P, P5
HALT
```

17-1-3 Using Shift Coordinates

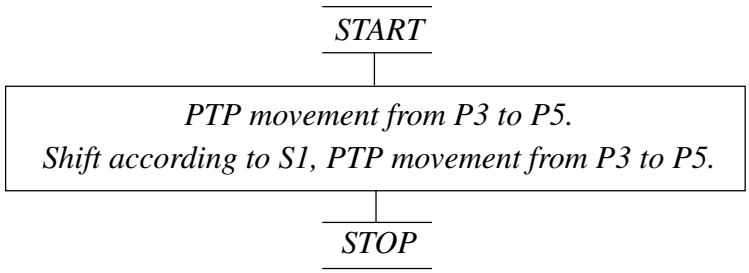
- *Outline*

As shown in the diagram, after PTP movement from P3 to P5, the coordinate system is shifted +100 along the X axis, -100 along the Y axis, and once again there is PTP movement from P3 to P5. The shift coordinate is set in S1 and P3, P4, P5 are set as describe in section 16-1-2.

S0=0.00 0.00 0.00 0.00
S1=100.00 -100.00 0.00 0.00



- *Process Flow*



- *Program Example*

```
SHIFT SO
FOR J=3 TO 5
   MOVE P, P[J]
NEXT J
SHIFT S1
FOR K=3 TO 5
   MOVE P, P[K]
NEXT K
HALT
```

17-1-4 Palletizing

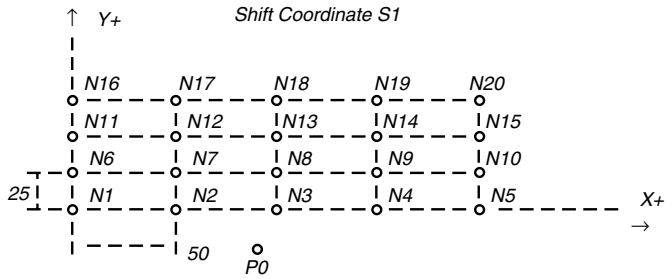
17-1-4-1 Utilization of the Shift Coordinates

- *Outline*

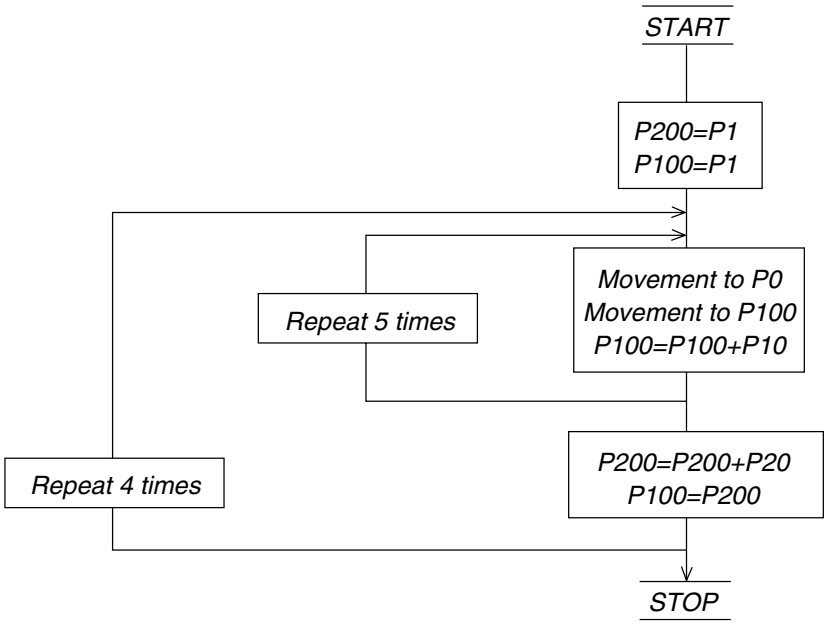
Movement between equidistant pallets and a fixed work supply position "P0" is performed with the following program. The points N1 to N20 are Cartesian coordinates. They are distanced from each other 50mm on the X axis, and 25mm on the Y axis. The arm moves from point to point P0-N1-P0-N2...N5-P0-N6-P0... while moving back and forth from point P0.

Work Supply Position: P0=		*0.0*	*0.0*	*0.0*	*0.0*	*0.0*
0.0						
X axis distance: P10=		*50.0*	*0.0*	*0.0*	*0.0*	*0.0*
0.0						
Y axis distance: P20=		*0.0*	*25.0*	*0.0*	*0.0*	*0.0*
0.0						
N1 position: P1=	*100.0*	*50.0*	*0.0*	*0.0*	*0.0*	*0.0*

The above is input in point data mode.



- *Process Flow*



- *Program Example*

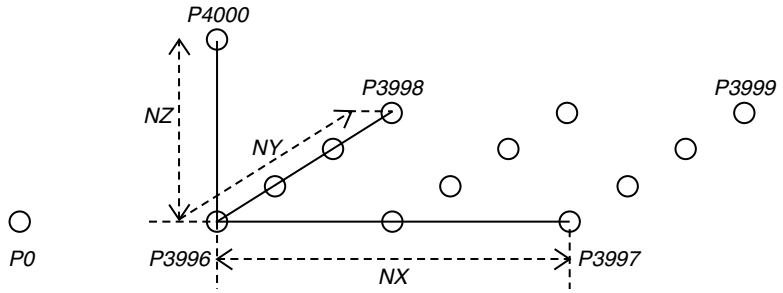
```
SHIFT S1
P100=P1
P200=P1
FOR J=1 TO 4
    FOR K=1 TO 5
     SHIFT S0
     MOVE P, P0
     SHIFT S1
     MOVE P, P100
     P100=P100+P10
    NEXT K
    P200=P200+P20
    P100=P200
NEXT J
HALT
```

17-1-4-2 Utilization of Palette Movement

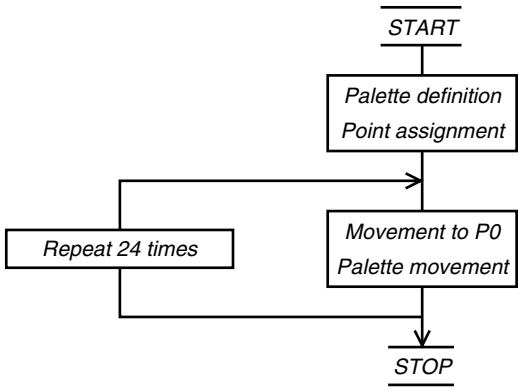
- *Outline*

Movement between pallets arranged an equal distance apart and a fixed work supply position "P0" is performed with the following program. The points N1 to N36 are Cartesian coordinates. They are in increment 50mm from each other on the X axis, 50mm on the Y axis, and 100mm on the Z axis. The arm moves back and forth between point P0 and each point as in the following sequence of P0-N1-P0-N2...N5-P0-N6... etc.

Work supply position : P0 = 0.00 0.00 100.00 0.00 0.00 0.00
Palette definition : PLT0 (usable point, P3996 to P4000)
NX= 3
NY= 4
NZ= 2
P3996=100.0050.00100.00 0.00 0.00 0.00
P3997=200.0050.00100.00 0.00 0.00 0.00
P3998=100.00200.00100.000.00 0.00 0.00
P3999=200.00200.00100.000.00 0.00 0.00
P4000=100.0050.00100.00 0.00 0.00 0.00



- *Process Flow*



- *Program Example*

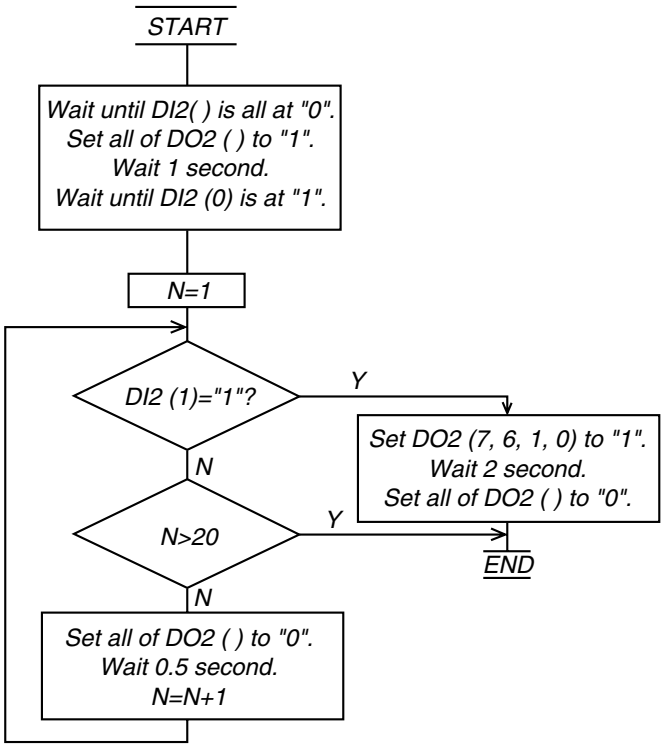
```
FOR I=1 TO 24
    MOVE P,P0,Z=0.00
    PMOVE(0,1),Z=0.00
NEXT I
MOVE P,P0,Z=0.00
HALT
```

17-1-5 DI/DO (Digital I/O) Movement

- *Outline*

Signal I/O with multi-purpose I/O devices.

- *Process Flow*



- *Program Example*

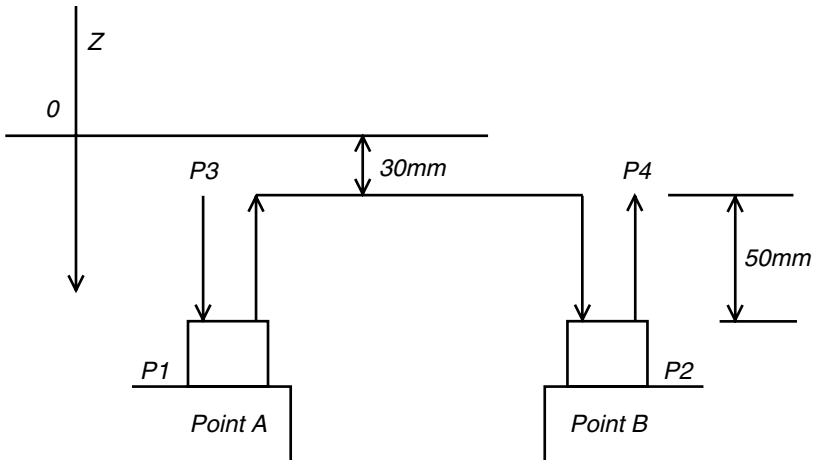
```
WAIT DI2()=0
DO2()=&B11111111
DELAY 1000
WAIT DI2(0)=1
N=1

*LOOP1: IF DI2(1)=1 THEN *PROGEND
IF N>20 THEN *ALLEND
DO2()=0
DELAY 500
N=N+1
GOTO *LOOP1

*PROGEND:
DO2(7, 6, 1, 0)=&B1111
DELAY 2000
DO2()=0
*ALLEND: HALT
```

17-2 Application

17-2-1 Pick and Place Between Two Points

● *Outline*

Grasp the part located at point A and place at point B.



● *Precondition*

1. *Set the robot movement path.*
   P3 → P1 → P3 → P4 → P2 → P4
   Locate P3 and P4 respectively at a position 50mm above P1 and P2 and set with P1 and P2 Teach.
2. *I/O Signal*
   D02(0) Chuck open/close= 0:OPEN and 1:CLOSE
   A 0.1 second wait time is set during chuck open and close.

● *Program Example*

① *For calculating to find P3 and P4*

```
P3=P1
P4=P2
LOCZ(P3)=LOCZ(P3)-50.0
LOCZ(P4)=LOCZ(P4)-50.0
MOVE P,P3
GOSUB *OPEN
MOVE P,P1
GOSUB *CLOSE
MOVE P,P3
MOVE P,P4
MOVE P,P2
GOSUB *OPEN
MOVE P,P4
HALT
*OPEN
DO2(0)=0
DELAY 100
RETURN
*CLOSE:
DO2(0)=1
DELAY 100
RETURN
```

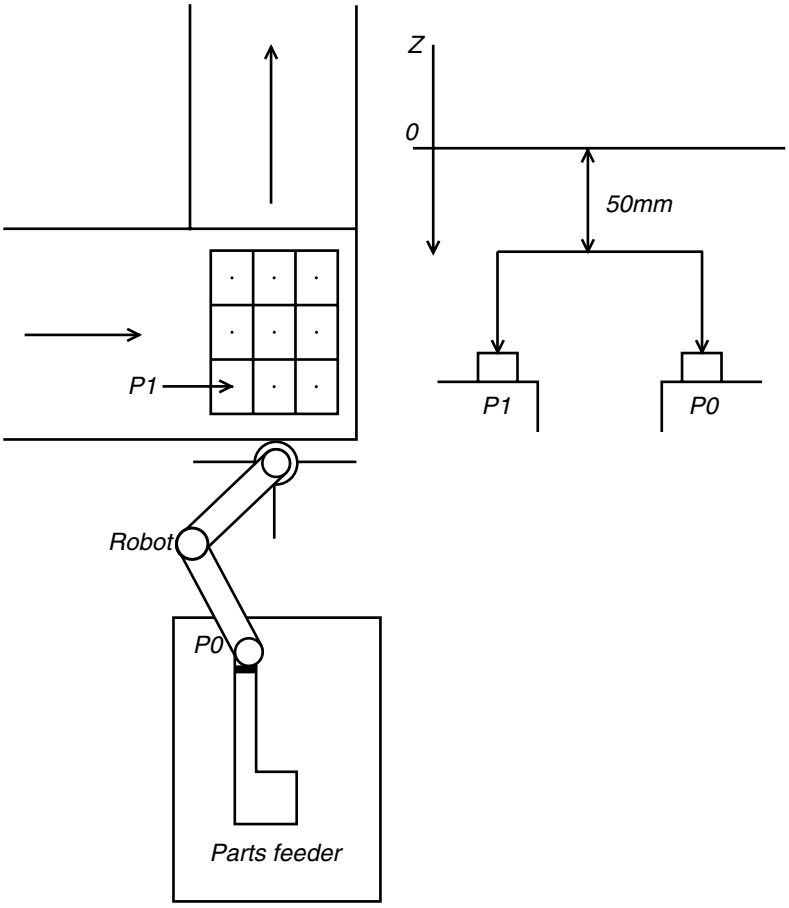
② *When using arch motion*

```
P4=P2
LOCZ(P4)=LOCZ(P4)-50.0
GOSUB *OPEN
MOVE P,P1,Z=30.0
GOSUB *CLOSE
MOVE P,P2,Z=30.0
GOSUB *OPEN
MOVE P,P4
HALT
*OPEN:
DO2(0)=0
DELAY 100
RETURN
*CLOSE:
DO2(0)=1
DELAY 100
RETURN
```

17-2-2 Palletizing

● *Outline*

Parts supplied from the parts feeder are loaded at fixed intervals onto the pallet. The pallet is ejected when full.



● *Precondition*

1. *I/O Signal*

DI(30)	=1:Parts supply
DI(31)	=1:Pallet

DO(30)	Robot hand open/close=0:OPEN and 1:CLOSE
DO(31)	=1:Pallet eject

Robot hand open and close time is 0.1 second and pallet eject time is 0.5 seconds.

2. *The points below are all input beforehand as point data.*

P0	Part supply position
P1	Pallet reference position
P10	X direction pitch
P11	Y direction pitch

3. *Move to a position Z=50mm above the pallet and parts feeder.*

- *Program Example 1*

```
WHILE -1
FOR A=0 TO 2
FOR B=0 TO 2
WAIT DI(31)=1
WAIT DI(30)=1
DO(30)=0
DELAY 100
MOVE P,P0,Z=50.0
DO(30)=1
DELAY 100
P100=P1+P10*B+P11*A
MOVE P,P100,Z=50.0
DO(30)=0
DELAY 100
NEXT
NEXT
DRIVE (3,0)
DO(31)=1
DELAY 500
DO(31)=0
WEND
HALT
```

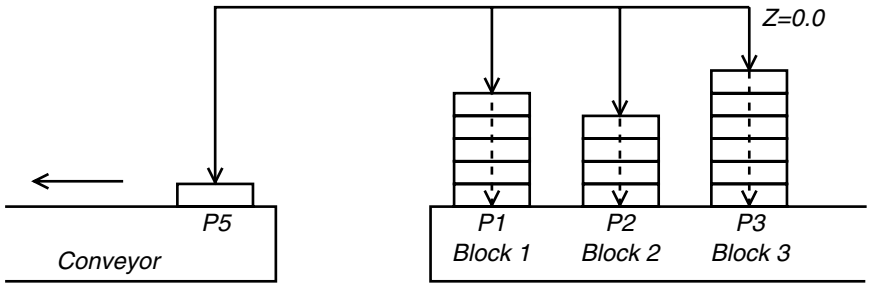
- *Program Example 2*

```
* When defined in pallet 0
WHILE -1
FOR A=1 TO 9
WAIT DI(31)=1
WAIT DI(30)=1
DO(30)=0
DELAY 100
MOVE P, P0, Z=50.0
DO(30)=1
DELAY 100
PMOVE (0, A), Z=50.0
DO(30)=1
DELAY 100
NEXT
DRIVE(3, 0)
DO(31)=1
DELAY 500
DO(31)=0
WEND
HALT
```

17-2-3 Pick and Place of Parts Stacked in Layers

● *Outline*

Parts stacked in a maximum of 6 layers and 3 blocks are grasped and placed at fixed intervals onto the conveyor. The number of parts used per block is optional. Part detection is done by means of a sensor installed on the robot hand.



● *Precondition*

1. *I/O signal*

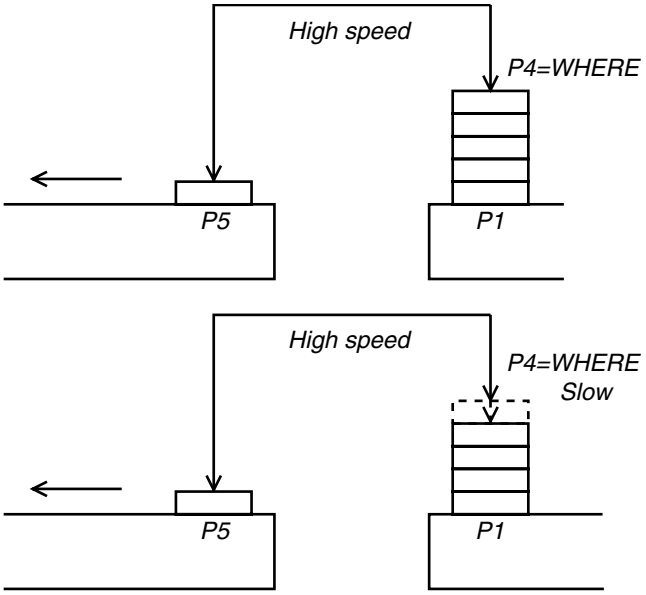
DI(30)	Part detection sensor=1:Parts
DO(30)	Robot hand open/close=0:OPEN and 1:CLOSE

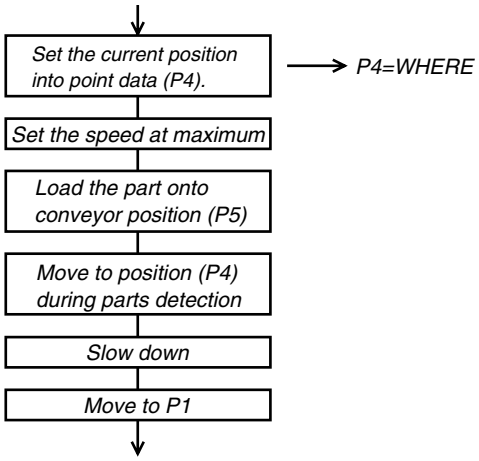
Robot hand open and close time is 0.1 seconds.

2. *The points below are all input beforehand as point data.*

P1	Bottom of block 1
P2	Bottom of block 2
P3	Bottom of block 3
P5	Position on conveyor

3. *During maximum speed movement, slow down when in proximity to the part.*




Set the current position into point data (P4).
P4=WHERE
Set the speed at maximum
Load the part onto conveyor position (P5)
Move to position (P4) during parts detection
Slow down
Move to P1

4. *Use the MOVE statement STOPON condition for movement during sensor detection.*

- *Program Example*

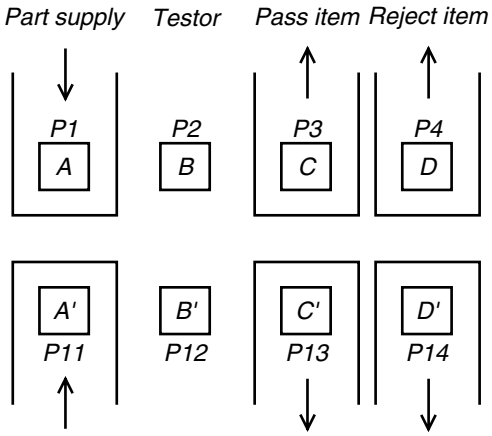
```
FOR A=1 TO 3
SPEED 100
GOSUB *OPEN
P6=P[A]
LOCZ(P6)=0.00
MOVE P,P6,Z=0.0
WHILE -1
SPEED 20
MOVE P,P[A],STOPON DI3(0)=1
IF DI3(0)=0 THEN *L1
'SENSOR ON
P4=JTOXY(WHERE)
GOSUB *CLOSE
SPEED 100
MOVE P,P5,Z=0.0
GOSUB *OPEN
MOVE P,P4,Z=0.0
WEND
*L1: 'SENSOR OFF
NEXT A
SPEED 100
DRIVE (3,0)
HALT
*OPEN:
DO3(0)=0
DELAY 100
RETURN
*CLOSE:
DO3(0)=1
DELAY 100
RETURN
```

17-2-4 Parts Inspection 1 (Multi-tasking Example)

● *Outline*

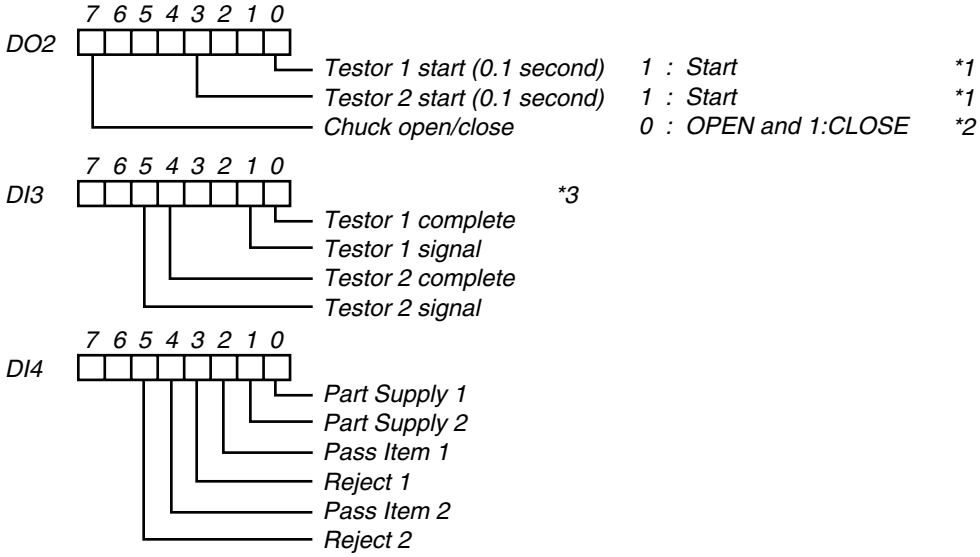
One robot is used to detect to test two different parts and make a pass/fail inspection. The part at point A is grasped and moved to the testor at point B. The testor makes a pass/fail test. Passed items are sent to point C and rejects are sent to point D.

In the same way parts at point A′ are grasped and moved to the testor at point B′. The testor makes a pass/fail test. Passed items are sent to point C ′and rejects are sent to point D′. 10 to 20 seconds are required for the pass/fail test by the testor.



● *Precondition*

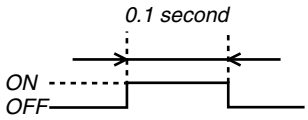
1. *I/O Signal*


7 6 5 4 3 2 1 0
DO2
Testor 1 start (0.1 second) 1 : Start *1
Testor 2 start (0.1 second) 1 : Start *1
Chuck open/close 0 : OPEN and 1:CLOSE *2
7 6 5 4 3 2 1 0
DI3
*3
Testor 1 complete
Testor 1 signal
Testor 2 complete
Testor 2 signal
7 6 5 4 3 2 1 0
DI4
Part Supply 1
Part Supply 2
Pass Item 1
Reject 1
Pass Item 2
Reject 2

POINT

**1 Set the start signal to apply a 0.1 second pulse signal to the testor.*

**2 Chuck open and close time is 0.1 seconds.*



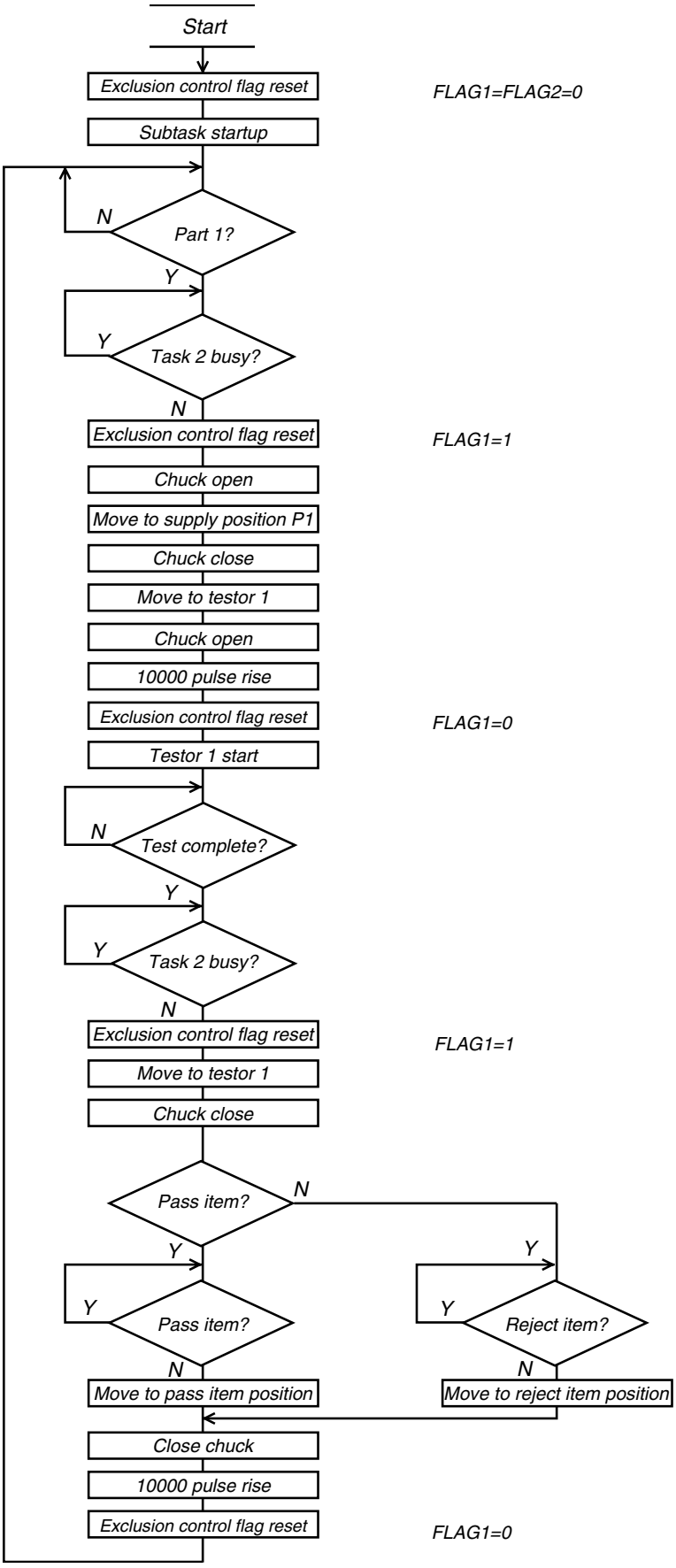
**3 Signals sent from the testor are completion and pass/fail item signals. After testing a completion signal ON (=1) is sent. A (1) is a pass item signal and a (0) is a reject item signal.*

2. *Startup the subtask and then control part 1 with the main task (task 1) and part 2 with the subtask (task 2).*

3. *The exclusion control flag is used so that other tasks can be run while waiting for the completion signal from the testor.*

   FLAG1 = 1 : Task 1 busy (Task 2 is kept waiting)
   = 0 : Task 1 ready (Task 2 in progress)
   FLAG2 = 1 : Task 2 busy (Task 1 is kept waiting)
   = 0 : Task 2 ready (Task 1 in progress)

4. Flow Chart



The task 2 (subtask) uses a similiar flowchart.

- *Program Example*

```
FLAG1=0
FLAG2=0
UPPOS=0.0
START *S1,T2                Subtask Startup
*L1:
WAIT DI4(0)=1               Part supply standby
WAIT FLAG2=0                Task complete standby
FLAG1=1                     Exclusion control flag reset
GOSUB *OPEN                 Chuck open
MOVE P,P1,Z=UPPOS           Move to supply position
GOSUB *CLOSE                Chuck close
MOVE P,P2,Z=UPPOS           Move to testor
GOSUB *OPEN                 Chuck open
DRIVEI (3,-10000)           Z 10,000 pulse rise
FLAG1=0                     Exclusion control flag reset
DO2(0)=1                    Testor start
DELAY 100
DO2(0)=0
WAIT DI3(0)=1               Test complete standby
WAIT FLAG2=0                Task complete standby
FLAG1=1                     Exclusion control flag reset
MOVE P,P2,Z=UPPOS           Move to testor
GOSUB *CLOSE                Chuck close
IF DI3(1)=1 THEN            Test
  'GOOD
  WAIT DI4(2)=0             Part movement standby
  MOVE P,P3,Z=UPPOS         Move to pass item position
ELSE
  'NG
  WAIT DI4(3)=0             Part movement standby
  MOVE P,P4,Z=UPPOS         Move to reject item position
ENDIF
GOSUB *OPEN                 Chuck open
DRIVEI (3,-10000)           Z 10,000 pulse rise
FLAG1=0                     Exclusion control flag reset
GOTO *L1
*OPEN:
DO2(7)=0
DELAY 100
RETURN
*CLOSE:
DO2(7)=1
DELAY 100
RETURN
*S1:
WAIT DI4(1)=1
WAIT FLAG1=0
FLAG2=1
GOSUB *OPEN
MOVE P,P11,Z=UPPOS
GOSUB *CLOSE
MOVE P,P12,Z=UPPOS
GOSUB *OPEN
DRIVEI (3,-10000)
FLAG2=0
DO2(3)=1
DELAY 100
DO2(3)=0
WAIT DI3(4)=1
WAIT FLAG1=0
FLAG2=1
MOVE P,P12,Z=UPPOS
GOSUB *CLOSE
IF DI3(5)=1 THEN
  'GOOD
  WAIT DI4(4)=0
  MOVE P,P13,Z=UPPOS
ELSE
  'NG
  WAIT DI4(5)=0
  MOVE P,P14,Z=UPPOS
ENDIF
GOSUB *OPEN
DRIVEI (3,-10000)
FLAG2=0
GOTO *S1
```

17-2-5 Parts Inspection 2 (2 Robots Example)

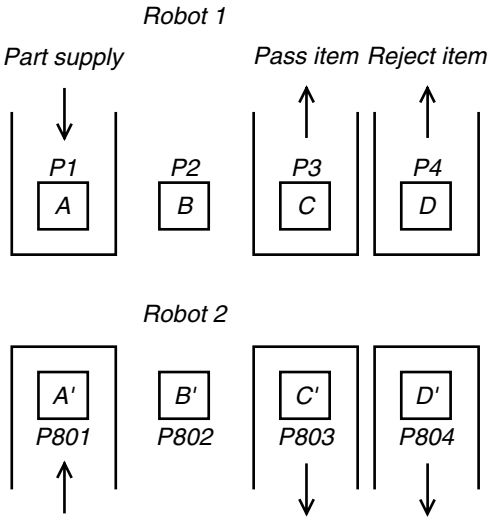
● *Outline*

The operation shown in "16-2-4 Parts Inspection 1" can be executed by using two robots.

Two robots are used with a testor to identify and test two different parts and make a pass/fail inspection.

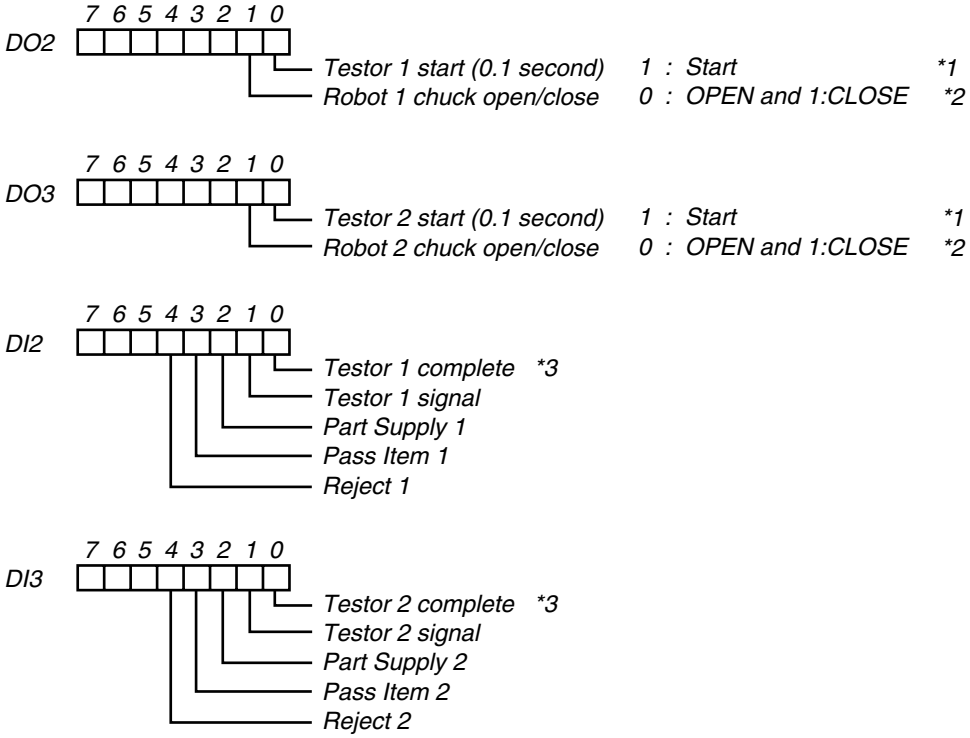
With robot 1, the part at point A is grasped and moved to the testor at point B. The testor makes a pass/fail test. Passed items are sent to point C and rejects are sent to point D.

In the same way, with the robot 2, parts at point A′ are grasped and moved to the testor at point B′. The testor makes a pass/fail test. Passed items are sent to point C′ and rejects are sent to point D′. Several seconds are required for the pass/fail test by the testor.



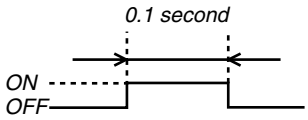
● *Precondition*

1. I/O Signal


7 6 5 4 3 2 1 0
DO2
Testor 1 start (0.1 second) 1 : Start *1
Robot 1 chuck open/close 0 : OPEN and 1:CLOSE *2
7 6 5 4 3 2 1 0
DO3
Testor 2 start (0.1 second) 1 : Start *1
Robot 2 chuck open/close 0 : OPEN and 1:CLOSE *2
7 6 5 4 3 2 1 0
DI2
Testor 1 complete *3
Testor 1 signal
Part Supply 1
Pass Item 1
Reject 1
7 6 5 4 3 2 1 0
DI3
Testor 2 complete *3
Testor 2 signal
Part Supply 2
Pass Item 2
Reject 2

POINT

**1 Set the start signal to apply a 0.1 second pulse signal to the testor.*



**2 Chuck open and close time is 0.1 seconds.*

**3 Signals sent from the testor are completion and pass/fail item signals. After testing a completion signal ON (=1) is sent. A (1) is a pass item signal and a (0) is a reject item signal.*

2. Startup the subtask and then control part 1 with the main task (task 1) and part 2 with the subtask (task 2).

3. The exclusion control flag is used so that other tasks can be run while waiting for the completion signal from the testor.

FLAG1 = 1 : Task 1 busy (Task 2 is kept waiting)
= 0 : Task 1 ready (Task 2 in progress)
FLAG2 = 1 : Task 2 busy (Task 1 is kept waiting)
= 0 : Task 2 ready (Task 1 in progress)

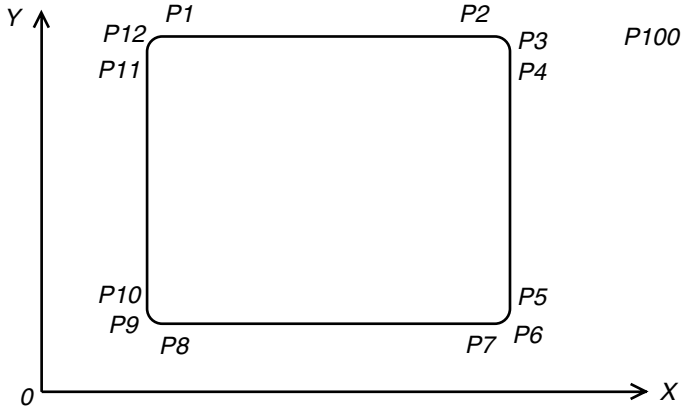
- *Program Example* *Description*

```
UPPOS=0.0
START *S1,T2                Subtask Startup
*L1:                                                    *S1:
 WAIT DI2(2)=1              Part supply standby         WAIT DI3(2)=1
 GOSUB *OPEN_MAIN           Chuck open                  GOSUB *OPEN_SUB
 MOVE P,P1,Z=UPPOS          Move to supply position     MOVE2 P,P801,Z=UPPOS
 GOSUB *CLOSE_MAIN          Chuck close                 GOSUB *CLOSE_SUB
 MOVE P,P2,Z=UPPOS          Move to testor              MOVE2 P,P802,Z=UPPOS
 GOSUB *OPEN_MAIN           Chuck open                  GOSUB *OPEN_SUB
 DRIVEI (3,-50.00)          Z 50.0mm rise               DRIVEI2 (3,-50.00)
 DO2(0)=1                   Testor start                DO3(0)=1
 DELAY 100                                              DELAY 100
 DO2(0)=0                                               DO3(0)=0
 WAIT DI2(0)=1              Test complete standby       WAIT DI3(0)=1
 MOVE P,P2                  Move to testor              MOVE2 P,P802
 GOSUB *CLOSE_MAIN          Chuck close                 GOSUB *CLOSE_SUB
 IF DI2(1)=1 THEN           Test                        IF DI3(1)=1 THEN
  'GOOD                                                  'GOOD
  WAIT DI2(3)=0             Pass item movement standby   WAIT DI3(3)=0
  MOVE P,P3,Z=UPPOS         Move to pass item position   MOVE2 P,P803,Z=UPPOS
 ELSE                                                   ELSE
  'NG                                                    'NG
  WAIT DI2(4)=0             Reject item movement standby WAIT DI3(4)=0
  MOVE P,P4,Z=UPPOS         Move to reject item position MOVE2 P,P804,Z=UPPOS
 ENDIF                                                  ENDIF
 GOSUB *OPEN_MAIN           Chuck open                  GOSUB *OPEN_SUB
 DRIVEI (3,-50.00)          Z 50.0mm rise               DRIVEI2 (3,-50.00)
 GOTO *L1                                               GOTO *S1
*OPEN_MAIN:                                            *OPEN_SUB:
 DO2(1)=0                                               DO3(1)=0
 DELAY 100                                              DELAY 100
 RETURN                                                 RETURN
*CLOSE_MAIN:                                           *CLOSE_SUB:
 DO2(1)=1                                               DO3(1)=1
 DELAY 100                                              DELAY 100
 RETURN                                                 RETURN
```

17-2-6 Sealing

● *Outline*

Carry out part sealing using the information below.



● *Precondition*

1. *I/O Signal*

DI3(0)	Sealing start signal=1:Start
DO3(0)	Valve open/close=1:OPEN and 0:CLOSE

2. *Set teach for P1 to P12 and P100 (waiting position)*

● *Program Example*

```
 P0=P1
  LOCZ(P0)=LOCZ(P1)-50.0
*L1:
  MOVE P,P100
  WAIT DI3(0)=1
  MOVE P,P0
  MOVE P,P1
  DO3(0)=1
  MOVE L,P2
  MOVE C,P3,P4
  MOVE L,P5
  MOVE C,P6,P7
  MOVE L,P8
  MOVE C,P9,P10
  MOVE L,P11
  MOVE C,P12,P1
  WAIT ARM
  DO3(0)=0
  MOVE P,P0
  GOTO *L1
```

18 Sequence Program

Besides regular robot programs, this controller is capable of executing high-speed processing programs (sequence programs) in response to the robot's output (DI/DO, MO). This means that the controller is capable of executing two different types of programs, the robot program and the sequence program, at the same time.
The sequence program is executed according to its own independent cycle, regardless of the execution or stopping of the robots program. The sequence program begins operation as soon as the controller is turned on (that is, in "MANUAL" mode). This means that it can be used to monitor the status of sensors, push button switches, electromagnetic valves, etc.
The sequence program is written in the same robot language used for robot programs, eliminating the need to learn a new language and making it easier to program.

18-1 Creating Sequence Programs

18-1-1 Programming Method

It is necessary to create sequence programs in order to take advantage of the sequencer's capabilities. First, put the controller in "PROGRAM" mode and create a file with the file name "SEQUENCE". With this name, this file will be recognized automatically by the controller as the sequence program.

```
PROGRAM >DIR                                    <TEST10   >
-------------------------------------------------------
 No. 1     NAME       LINE       BYTE     RW/RO
  1     TEST10          12        145        RW
  2     LOCATE20        25        320        RW


Enter program name >SEQUENCE
```

Fig. 18-1-1-1 Sequence program file creation

Next the program is input. Create the program in the same way as you would create a robot program (refer to the separate "10-2 Program Editing" in Chapter 4 on the User's Manual). The commands that may be used are discussed in section "18-3 Programming the Sequencer".

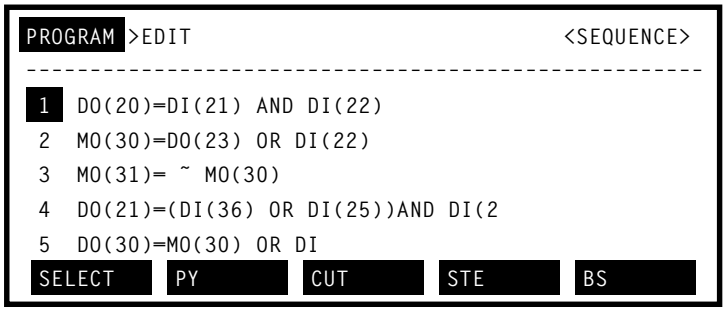

PROGRAM >EDIT <SEQUENCE>
1 DO(20)=DI(21) AND DI(22)
2 MO(30)=DO(23) OR DI(22)
3 MO(31)= ~ MO(30)
4 DO(21)=(DI(36) OR DI(25))AND DI(2
5 DO(30)=MO(30) OR DI
SELECT PY CUT STE BS

Fig. 18-1-1-2 Inputting program lines

18-1-2 Compiling

After editing the program, compiling is done for the sequencer. Compiling is performed in the same way as for robot programs — press the key with the screen of the top position of "PROGRAM" mode (Fig. 18-1-2-1).

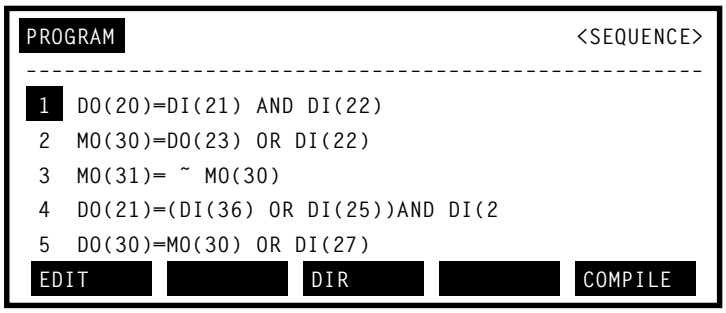


Fig. 18-1-2-1 Compiling sequence program

A message will be displayed asking if you want to compile the program or not. Press the F 4 *key and the program will be compiled. Press the* F 5 *key and the program will not be compiled, the controller will display the normal process screen for robot program compiling.*
When the F 4 *key is pressed the sequence program will be compiled (Fig. 18-1-2-2).*

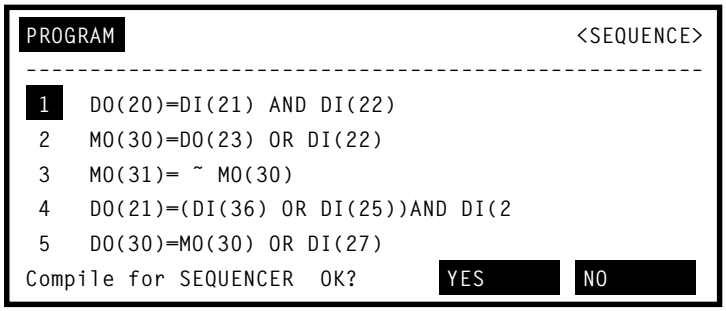

PROGRAM <SEQUENCE>
1 DO(20)=DI(21) AND DI(22)
2 MO(30)=DO(23) OR DI(22)
3 MO(31)= ~ MO(30)
4 DO(21)=(DI(36) OR DI(25))AND DI(2
5 DO(30)=MO(30) OR DI(27)
Compile for SEQUENCER OK? YES NO

Fig. 18-1-2-2 Sequence program compile check

If there is an error in the program an error message will be displayed, and the program will be listed from the line with the error (Fig. 18-1-2-3)
If there are no bugs in the program the program will be listed from its first line.

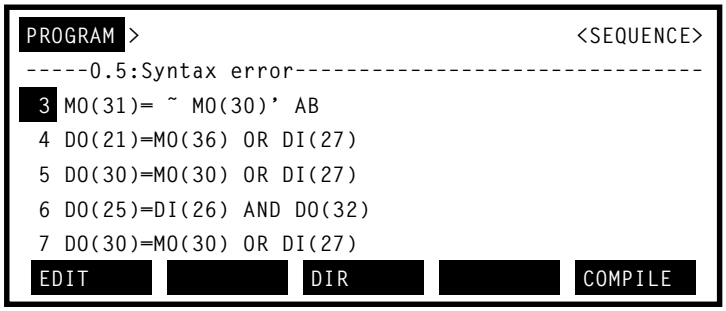


Fig. 18-1-2-3 Error during compiling

Also, when compiling ends without any errors, and the directory is called up to the screen, the program name "SEQUENCE" will appear with a lower case "s" before it. This shows that the sequence program has been compiled successfully and is ready for use (Fig. 18-1-2-4).

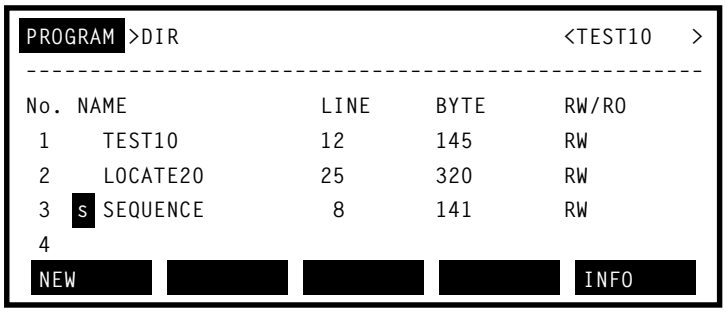


Fig. 18-1-2-4 Sequence program compiled and stored

CAUTION

The sequence program is erased and the "s" is no longer displayed in the following cases. In these cases the sequencer may not be used in "UTILITY" mode.

1. ***When the sequence program is erased***
2. ***When the sequence program has been edited***
3. ***When a regular robot program has been compiled instead of the sequence program***

18-2 Running Sequence Programs

The conditions for running sequence programs are as shown below. Sequence programs may not be run unless all of the conditions are met.

1. *The sequence program must input and compiled to be readied for execution.*
2. *The sequencer must be enabled in "UTILITY" mode (refer to the separate "14-2 Permitting/Prohibiting Sequencer Execution" in Chapter 4 on the User's Manual).*
3. *The sequence control input port (DI (10)) must be closed.*
4. *The sequencer may be run only in "MANUAL" or "AUTO" mode.*

When all of the above conditions are met the sequence program may be run. While the program is running, the "s" will appear at the left end of the second line of the screen. (Fig. 18-2-1)

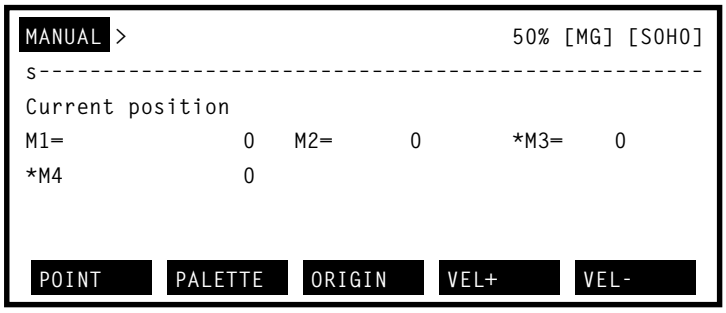


Fig. 18-2-1 Sequence program is running

18-2-1 Sequence Program Step Running

The sequence program may be run line by line with the STEP feature. In this way the movement of the robot can be verified carefully.
Press the F 5 *key when the compile screen is displayed (Fig. 18-1-2-2) , the screen is cancelled and the normal robot compile screen (Fig. 18-2-2) is displayed. Press the* F 4 *key and create a normal program to be executed. In "AUTO" mode, execute the program with the STEP feature and check its movement.*

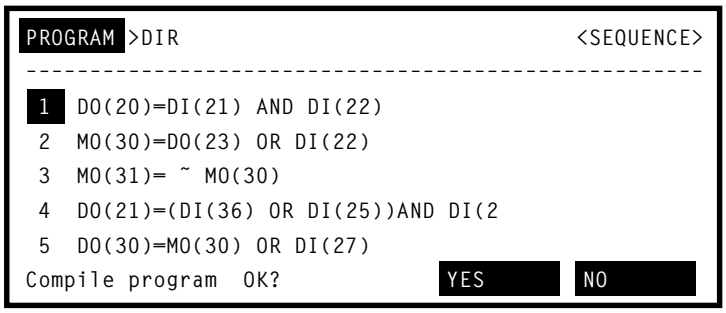


Fig. 18-2-2 Sequence program STEP execution

18-3 Programming the Sequencer

Sequencer program is created only with expressions composed of I/O variables and logical operators. An error will result during compiling if any statements other than these expressions is used in the program. Compiling will not be completed.

18-3-1 Assignation Statements that May be Used with the Sequencer

<output variable> <internal auxiliary output variable> <arm lock output variable> <timer output variable>	=<expression>

<expression> may only be a logical operation between input/output variables or internal auxiliary output variables or arm lock output variables or timer output variables.

18-3-2 Input/Output Variables that May be Used with the Sequencer

a. Input Variables

These variables show the status of the input signal.

DI(mb)	m: Port number 0 to 7, 10 to 13 b: Bit definition 0 to 7

b. Output Variables

These variables refer to or define the status of the output signal.

DO(mb)	m: Port number 0 to 7, 10 to 11 b: Bit definition 0 to 7

Output to ports 0 and 1 is not possible.
However, DO(27) cannot be used for the QRC controller as it is already used as a custom output.

c. Internal Output Variables

These variables are output within the controller and are not output externally.

MO(mb)	*m: Port number 0 to 7, 10 to 13* *b: Bit definition 0 to 7*

Output to ports 0 and 1 is not possible.

d. Arm Lock Output Variables

These variables are used to prohibit the arm movement. An axis movement is prohibited by ON status.

LO(mb)	*m: Port number 0* *b: Bit definition 0 to 7*

LO(00) to LO(07) corresponds to the 1st arm to the 8th arm.

e. Timer Output Variables

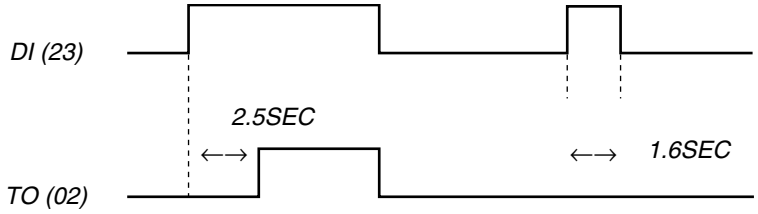
TO(mb)	*m: Port number 0* *b: Bit definition 0 to 7*

There are a total of 8 timer output variables: TO(00) to TO(07). The timer of each variable is defined by the timer definition statement TIM00 to TIM07.

USING TIMER EXAMPLE:

```
T I M 0 2 = 2 5 0 0
T O ( 0 2 ) = D I ( 2 3 )
```

When DI(23) is ON, after 2.5 seconds, TO(02) is set ON.
When DI(23) is OFF, TO(02) is also OFF.
When DI(23) isn't ON after 2.5 second or more, TO(02) does not change to ON.



Variables may only be defined in the 1 bit format.

EXAMPLES:

```
D O ( 3 5 )
M O ( 2 4 )
D I ( 1 6 )
```

Incorrect Examples:

```
D O ( 3 7 ,  2 4 )
D I 3 ( 4 )
M O 3 ( )
```

18-3-3 Timer Definition Statements

These variables show the status of the input signal.

TIMmb=<expression>	m: Port number 0 b: Bit definition 0 to 7

The value of <expression> must be from 100 to 999900msec (=0.1 second to 999.9 second).
However, since the unit is set for every 100msec, values less than 99msec are eliminated.

EXAMPLES:

```
T I M 0 0 = 6 5 0 ---------- 0.6 seconds
T I M 0 3 = 2 4 8 0 ---------- 2.4 seconds
```

Timer definition statements set the timer value of the timer output variable. This definition statement may be anywhere on the program. When the timer definition statement is omitted, the timer setting value of the variable is 0.
TIM00 to 07 correspond to the timer output variables TO(00) to (07).

18-3-4 Arithmetical Functions (Logical Operators) Used with the Sequencer

OR, |
AND, &
NOT, ~

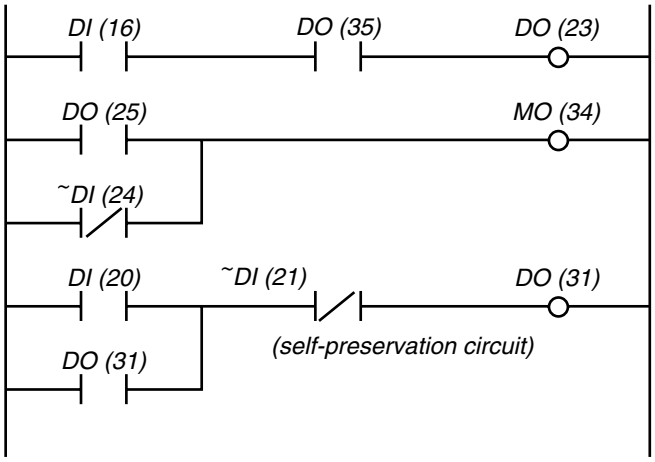
18-3-5 Priority of Logical Operations

1. *Expressions in parentheses*
2. *NOT, ~*
3. *AND, &*
4. *OR, |*

Example of Assignation:
For reference the exchange of a ladder diagram is shown.

```
DO(23) = DI(16) & DO(35)
MO(34) = DO(25) | ~DI(24)
DO(31) = (DI(20) | DO(31)) & ~DI(21)
```

Ladder Diagram



CAUTION

1. ***NOT must be used directly in front of the first parenthesis or directly to the left of an expression. The following commands are not possible:***
   DO(21)= ˜ (DI(30) | DI(32))
   ˜ DO(30)=DI(22) & DI(27)
2. ***Numerals may not be used to assign values on the right of the expression. The following examples may not be used:***
   MO(35)=1
   DO(26)=0
3. ***There is no need to place "HALT" or "HOLD" statements at the end of the program.***
4. ***The I/O and internal auxiliary output variables are common to variables used in robot programs so be careful not to use a variable name already occurring in other programs.***

Sequencer Specifications

Commands	AND, OR, NOT
I/O	Same as Robot Language
Program Storage	2048 bytes (maximum 250 steps)
Scan Time	From minimum of 10 msecs. to maximum of 100 msecs. depending on number of steps, changing automatically.

Appendix

A. Reserved Word List

ABSRST	ENDIF	NEXT	SHIFT
ABOVE	EOF	NONFLIP	SHIFT2
ABS	ERA	NOT	SIN
ACC	ERL	OFF	SKIP
ACCEL	ERR	OFFLINE	SPEED
ACCEL2	ERROR	ON	SPEED2
ACCESS	EXIT	ONLINE	SPN
ALL	FDD	OR	SQR
AND	FLIP	ORD	SRUN
ARCH	FN	ORGORD	START
ARCH2	FOR	ORGORD2	STEP
ARM	FREE	ORIGIN	STOP
ARM2	FUNCTION	OUT	STOPON
ARMTYPE	GASP	OUTPOS	STR
ARMTYPE2	GO	OUTPOS2	SUB
ARY	GOHOME	P	SUSPEND
ASC	GOSUB	PALET	SWI
ASPEED	GOTO	PASS	SYS
ASPEED2	HALT	PADDR	SYSTEM
ATN	HAND	PDEF	TAN
ATN2	HAND2	PLN	TASK
ATTR	HEX	PLT	TCOUNTER
AUTO	HND	PMOVE	THEN
AXWGHT	HOME	PMOVE2	TIME
AXWGHT2	HOLD	PPNT	TIMER
BELOW	IF	PRINT	TO
BIN	IN	PROGRAM	TOLE
BIT	INIT	PGM	TOLE2
BREAK	INPUT	PNT	TORQUE
BYTE	INT	POS	TORQUE2
CALL	INTEGER	PRM	UNIT
CASE	IRET	PTP	UNTIL
CHANGE	JTOXY	PWR	VAL
CHANGE2	JTOXY2	RADDEG	VAR
CHGTSK	KEY	READ	VCAL
CHR	LEFT	REF	VDEF
CMU	LEFTY	REN	VEL
CONFIG	LEFTY2	RELESE	VER
COO	LEN	REM	VFEA
COPY	LET	REMOTE	VFILE
COS	LOCA	RESET	VGET
CUT	LOCB	RESTART	VIS
DATE	LOCR	RESUME	VNAME
DEC	LOCX	RETURN	VOPT
DECLARE	LOCY	RIGHT	VSET
DEF	LOCZ	RIGHTY	WAIT
DEFIO	LOOP	RIGHTY2	WEIGHT
DEFPOS	LSHIFT	RO	WEIGHT2
DEGRAD	MANUAL	ROTATE	WEND
DELAY	MCHREF	RSHIFT	WHERE
DI	MCHREF2	RUN	WHERE2
DIM	MEM	RW	WHILE
DIO	MID	S	WHRXY
DIR	MIRROR	SCR	WHRXY2
DIS	MOD	SELECT	WORD
DIST	MOVE	SEND	WRITE
DO	MOVE2	SEQUENCE	XOR
DRIVE	MOVEI	SERVO	XY
DRIVE2	MOVEI2	SERVO2	XYTOJ
DRIVEI	MRF	SET	XYTOJ2
DRIVEI2	MSG	SFT	YZ
ELSE	MSPEED	SGI	ZX
ENB	MSPEED2	SGR	
END	NAME	SHARED	

NOTE

Since the above are reserved as robot language items, they cannot be used as identifiers (variables etc.).

YAMAHA ROBOT CONTROLLER
MRC Series
QRC Series
MRCH Series
QRCH Series
Programming MANUAL

July. 2002, 13th Edition
© YAMAHA MOTOR CO., LTD.
IM Company

All rights reserved. No part of this publication
may be reproduced in any form without the
permission of YAMAHA MOTOR CO., LTD.